UNIVERSITÀ DI PISA

SCUOLA DI INGEGNERIA

CORSO DI LAUREA MAGISTRALE IN INGEGNERIA INFORMATICA

Progettazione e realizzazione della logica applicativa di sistemi di monitoraggio ambientale: un approccio basato su Business Rule Management System

Candidato:

Antonino Cosenza

Relatori:

Prof. Gigliola Vaglini

Prof. Mario Giovanni C.A. Cimino

Anno Accademico 2011/2012

Indice

Sommario

Il lavoro svolto riguarda l'analisi, la progettazione e l'implementazione di un sistema di monitoraggio e controllo di reti di sensori che misurano dati ambientali e meteorologici. La prima fase ha riguardato l'analisi dell'infrastruttura di acquisizione dati. A partire da tale piattaforma hardware e software, è stato realizzato un livello software in grado di monitorare eventi per eseguire monitoraggio ambientale oppure di calcolare degli indicatori di medio-lungo termine, al fine di rilevare condizioni ambientali anomale. Per la realizzazione della logica di monitoraggio e controllo si è scelto di utilizzare un Business Rule Management System, al fine di garantire un'alta manutenibilità e indipendenza dalla piattaforma tecnologica.

Introduzione

Un BRMS o Business Rule Management System è un sistema software che permette di definire, distribuire, eseguire, monitorare e mantenere la varietà e la complessità della logica decisionale utilizzata dai sistemi presenti all'interno di un'organizzazione o impresa. Questa logica, nota anche come regole di business, comprende le politiche, i requisiti e le istruzioni condizionali utilizzati per determinare le azioni tattiche che si svolgono in applicazioni e sistemi.

Un sistema BRMS (Business Rule Management System) fornisce i criteri organizzativi e le decisioni operative associate a tali criteri, da definire, distribuire, monitorare e gestire separatamente dalle applicazioni cruciali. Esternalizzando le regole di business e fornendo strumenti per gestirle, una soluzione BRMS consente agli esperti di business di definire e gestire le decisioni che guidano il comportamento dei sistemi, riducendo le attività e il tempo necessari per aggiornarli e aumentando la possibilità di rispondere alle modifiche degli ambienti di business da parte delle aziende.

L'obiettivo del lavoro di tesi è la realizzazione di un prodotto software di elaborazione di campioni, che astragga sia dall'infrastruttura sottostante di lettura e archiviazione dei dati, sia da quella sovrastante di presentazione dei risultati dell'elaborazione stessa.

A fine di sfruttare al meglio la piattaforma hardware di raccolta ed esposizione dei dati, risulta fondamentale l'utilizzo di un idoneo sistema informativo atto a fornire gli strumenti necessari al coordinamento e al monitoraggio dei dati forniti e della stessa infrastruttura hardware.

In uno scenario di questo tipo si colloca perfettamente l'utilizzo di un Business Rule Management System. Tale sistema software consente infatti

l'implementazione, l'esecuzione e il mantenimento della business logic in un modulo software estraneo ai sistemi che la utilizzano.

Il nostro use case reale consiste in un'infrastruttura esistente di reti di sensori, i quali misurano dati metereologici e ambientali. La logica da implementare si articola attraverso diversi livelli applicativi di controllo e monitoraggio sia dei campioni raccolti che della stessa rete di sensori.

Tale business logic può quindi essere suddivisa nel modo seguente:

- Controllo e monitoraggio delle condizioni ambientali e metereologiche nel breve periodo;
- Creazione e aggiornamento di strutture dati statistiche relative alle metriche misurate;
- Controllo e monitoraggio delle condizioni ambientali e metereologiche nel lungo periodo;
- Controllo del corretto funzionamento dell'infrastruttura di rete.

Il lavoro di tesi svolto ha riguardato l'analisi di tale business logic e la progettazione e realizzazione di classi di regole in grado di implementarla.

Capitolo 1: Requisiti del sistema

L'obiettivo del software è quello di offrire uno strumento semplice e intuitivo per la gestione remota completa di una rete di monitoraggio ambientale.

Il sistema introduce la possibilità di effettuare un continuo monitoraggio real-time dei dati ambientali di una specifica area di interesse da remoto, senza quindi più la necessità di recarsi fisicamente presso la centrale di campionamento.

1.1. Il software

Il software da implementare si occupa di prelevare i nuovi dati con un periodo temporale configurato e processarli secondo la business logic definita, svolgendo le azioni previste.

In figura 1.1 viene mostrato lo schema a blocchi dell'architettura logica del sistema progettato.

Figura 1.1: Architettura logica del sistema

La rete di sensori rende disponibili a runtime i valori campionati tramite un formato xml predefinito, il quale riporta un timestamp, la tipologia del messaggio e, in base a questo, i valori delle metriche o informazioni di localizzazione.

Ai fini della sicurezza e riutilizzo futuro di tali campioni, questi sono memorizzati su un database a partire dal quale è possibile recuperarli tramite precise API (Application Programming Interface).

In figura 1.2 è riportata una rappresentazione grafica dell'infrastruttura hardware e software responsabile del campionamento e archiviazione sicura su database dei dati di interesse.

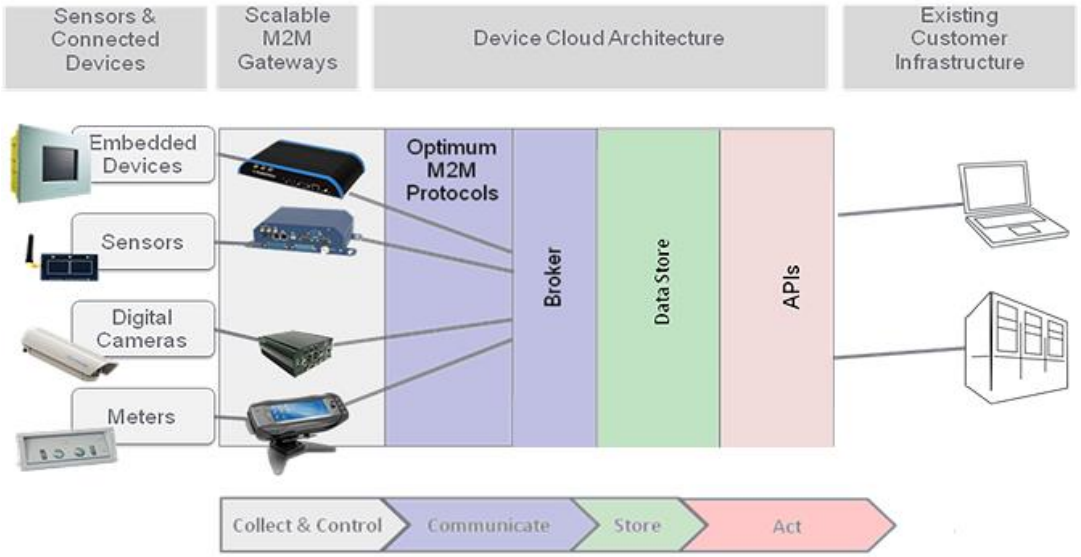

Figura 1.2: Architettura del sistema di campionamento

Attualmente non è presente un sistema software in grado di utilizzare le API esposte dall'infrastruttura di raccolta e archiviazione dei campioni e automatizzare il processo di analisi e verifica dei dati raccolti.

L'obiettivo del lavoro di tesi è quindi la realizzazione di un ulteriore livello software che si interfacci con l'infrastruttura esistente e sia in grado di:

- Rilevare eventi ambientali eccezionali in tempo reale;
- Calcolare degli indicatori di medio-lungo termine;
- Sfruttare tali indicatori al fine di rilevare condizioni ambientali anomale nel medio-lungo periodo;
- Monitorare il corretto funzionamento della piattaforma hardware.

Tale livello software è rappresentato dal *Business Data/Logic Integration and Monitoring Center*.

1.2. Business Data/Logic Integration and Monitoring Center

I requisiti funzionali di tale modulo software sono:

- Interfacciarsi con la piattaforma di raccolta dei dati tramite l'utilizzo delle API esposte, prelevando a intervalli di tempo regolari nuovi campioni;
- Elaborare tali campioni sulla base della business logic prevista;
- In seguito al verificarsi di un evento anomalo eseguire le procedure di emergenza definite dalla logica di business.

Per quanto riguarda i requisiti non funzionali, essi sono i seguenti:

- Al fine di aumentare la manutenibilità della business logic, questa deve essere tenuta separata da qualunque altro modulo dell'architettura del sistema;
- Al fine di garantire il riutilizzo di tale modulo software in contesti diversi, è necessario realizzare una netta separazione rispetto ai componenti responsabili della raccolta e presentazione sia dei campioni che dei risultati dell'elaborazione;
- La business logic deve essere "Cross Domain", quindi indipendente dalla macchina fisica o sistema operativo;
- Deve essere reso possibile il riutilizzo di regole pre-esistenti;
- Il sistema di archiviazione della business logic deve essere indipendente dai componenti che la utilizzano, affinché le

operazioni di aggiornamento e modifica non blocchino l'esecuzione di tali componenti;

- Al fine di rendere il software flessibile e adattabile e consentire possibili modifiche future da parte di personale diverso, le regole di business e flussi di elaborazione implementati devono essere di semplice comprensione;
- Diversi stakeholders (analisti, esperti di business, sviluppatori, etc) devono poter contribuire all'implementazione della logica di business, sulla base delle loro competenze: codice, Spreadsheet o Guided Editor.

1.3. Business Logic

La Business Logic da implementare può essere divisa in due blocchi principali:

- Controllo e monitoraggio delle metriche ambientali campionate al fine di rilevare eventi eccezionali sia nel breve che nel lungo periodo;
- Rilevazione dei malfunzionamenti hardware dell'infrastruttura sottostante.

1.3.1. Controllo e monitoraggio ambientale

La Business Logic da implementare relativa alla verifica delle condizioni ambientali, si divide a sua volta in due blocchi a seconda dell'intervallo temporale di monitoraggio: a breve e a lungo termine.

Per quanto riguarda il breve periodo, il software deve rilevare condizioni ambientali anomale verificatesi in un piccolo lasso temporale, ad

esempio nell'ultimo minuto. Tra tali condizioni rientrano sicuramente: il numero di particelle di particolato nell'aria supera la soglia di tolleranza; la forza media del campo elettromagnetico aumenta inaspettatamente; si registra un aumento della temperatura e contemporaneamente una diminuzione dell'umidità, indice di un incendio; ecc.

In particolare, nel breve periodo, possiamo individuare le seguenti classi di condizioni di anomalia:

- Ciascun valore campionato deve rientrare in un determinato intervallo sulla base della metrica di appartenenza e dell'area di installazione della rete di sensori. Al di fuori di tale intervallo, sia superiormente che inferiormente, è necessario verificare il corretto funzionamento del dispositivo e che non si siano presentate condizioni anomale;
- Il variare simultaneo di più metriche di tipo differente può essere indice del verificarsi di particolari eventi. E' quindi necessario monitorare contemporaneamente i valori registrati per ognuna di tali metriche;
- Aumenti considerevoli dei valori campionati per una data metrica in un corto lasso temporale;
- Le soglie minime e massime associate ad alcune metriche possono variare sulla base dei campioni misurati per altri parametri secondo una precisa relazione. E' quindi necessario monitorare i valori assunti da tali parametri, al fine di modificare in tempo reale le soglie dipendenti.

Per quanto riguarda invece il monitoraggio nel lungo periodo, questo consiste in un'attività di reportistica tesa ad aggregare i dati in ingresso ottenendo statistiche su intervalli temporali diversi rispetto al momento

corrente. Sono quindi previste statistiche su base oraria, giornaliera, settimanale, mensile. Mettendole in relazione è poi possibile individuare peggioramenti delle condizioni ambientali generali e intervenire rapidamente adottando misure preventive adeguate. Un esempio di intervento a lungo termine è rappresentato dall'adozione di zone a traffico limitato in quelle aree con un'alta umidità e concentrazione di CO2 nell'aria.

In particolare, nel lungo periodo, possiamo individuare le seguenti classi di condizioni di anomalia:

- Anche nel lungo periodo i valori medi, minimi e massimi registrati devono essere monitorati al fine di rilevare un peggioramento o miglioramento complessivo delle condizioni ambientali;
- I valori medi assunti da una determinata metrica in un determinato lasso temporale possono differire sensibilmente rispetto a quelli calcolati su un intervallo temporale più lungo. Questo può essere indice del verificarsi di eventi anomali. E quindi necessario monitorare la relazione tra i valori medi nel breve e nel lungo periodo.
- Anche nel lungo termine possono esistere relazioni tra le varie metriche per cui i valori medi assunti da alcune vanno a modificare le soglie di altre. Anche in questo caso è opportuno implementare opportuni controlli al fine di adeguare dinamicamente la business logic.

In seguito al rilevamento di una qualsiasi delle condizioni anomale sopra citate è previsto l'invio di un'opportuna notifica alle autorità competenti, che si faranno carico di esaminare le segnalazioni ed eseguire le azioni di emergenza previste.

1.3.2. Monitoraggio dell'infrastruttura hardware

Accanto alla logica di business vera e propria è prevista anche una logica di controllo dell'infrastruttura in grado di rilevare e segnalare eventuali problemi hardware occorsi: un dispositivo si disconnette inaspettatamente e non si riconnette in un tempo utile; non sono pubblicati nuovi dati di una data metrica per un lungo periodo di tempo, ecc.

In particolare, nel breve periodo, possiamo individuare le seguenti classi di condizioni di anomalia:

- Un dispositivo si disconnette inaspettatamente e non si riconnette in un tempo utile prefissato;
- Un dispositivo non pubblica valori relativamente a una data metrica per un determinato intervallo temporale. Anche in questo caso è previsto l'invio di un'opportuna segnalazione alle autorità competenti.

1.4. Analisi dei dati

Al fine di garantire un completo e soddisfacente monitoraggio ambientale, ogni sensore pubblica a determinati intervalli temporali due tipologie di messaggi:

- Una riportante i valori delle metriche misurate;
- L'altra riportante informazioni circa la localizzazione del sensore.

I due tipi di messaggi sono distinti dal campo "topic".

Come detto precedentemente, con l'obiettivo di permettere un monitoraggio completo dell'ambiente circostante, ogni sensore effettua campionamenti di diverse quantità.

Accanto a ogni messaggio contenente i campioni misurati nell'ultimo minuto, ogni sensore invia un secondo messaggio contenente tutte le informazioni necessarie circa la sua posizione.

Questi messaggi saranno ovviamente utilizzati per geolocalizzare l'area in cui si è verificata l'anomalia (ad esempio lo scoppio di un incendio indicato dall'aumento anomalo della temperatura e diminuzione dell'umidità) e intervenire nel minore tempo possibile.

In ogni messaggio, oltre al contenuto informativo oggetto di studio, sono presenti altri parametri complementari, utili in fase di analisi.

La struttura dati utilizzata dai dispositivi per l'esposizione dei dati al mondo esterno si articola attraverso tre blocchi, il cui contenuto informativo può variare sulla base della tipologia del messaggio:

- Il payload;
- Il timestamp dell'istante di ricezione e archiviazione del messaggio su database;
- Il topic su cui il messaggio è stato pubblicato.

Di seguito un'analisi più approfondita dei tre gruppi di informazione.

a) Payload

Il *payload* costituisce sicuramente il blocco principale del messaggio.

Nel caso dei messaggi relativi alle metriche, esso contiene due tipologie di informazioni:

- I valori delle metriche misurate;
- Il timestamp dell'istante in cui il messaggio è stato generato e inviato dal sensore.

Ogni messaggio riporta nel suo payload un campionamento completo delle metriche oggetto di studio.

Ogni campo "metric" segue il formato <Name,Type,Value> e offre tutte le informazioni necessarie ad una corretta analisi del campione corrispondente.

Anche nel caso dei messaggi riportanti informazioni di localizzazione dei sensori, sono presenti due tipologie di informazioni:

- Tutte quelle necessarie a individuare correttamente e con precisione la posizione del sensore;
- Un timestamp dell'instante in cui il messaggio è stato generato e inviato dal sensore.

Ogni messaggio riporta nel suo payload il set completo delle informazioni geografiche.

b) Timestamp

Subito dopo il payload, è riportato il *timestamp* relativo all'istante in cui il messaggio è stato ricevuto e memorizzato su database.

Si tratta dell'informazione temporale più importante dell'intero messaggio. Infatti è questo l'istante da utilizzare come riferimento nell'implementazione sia della Business Logic che della logica di controllo.

c) Topic

Il *topic* è utilizzato per stabilire la tipologia del messaggio e quindi il suo contenuto.

In particolare la stringa contenuta in questo campo, segue una specifica semantica e porta con sé una serie di informazioni.

In essa sono infatti riportate i tre seguenti parametri:

- Il primo, è il nome dell'account utente che si è autenticato presso il sistema con l'obiettivo di recuperare i nuovi campioni;
- Il secondo, è l'id del dispositivo dell'infrastruttura di raccolta dei dati che ha effettuato il campionamento;
- Il terzo, è l'informazione principale del blocco. E' questo infatti il parametro che consente di distinguere tra le due tipologie di messaggio.

Capitolo 2: Studio di Fattibilità

2.1. Analisi contesto e situazione attuale

Allo stato attuale il sistema è costituito da un infrastruttura di raccolta e archiviazione dei dati basata sull'utilizzo di reti di sensori.

Questa infrastruttura hardware, già esistente e funzionante, consente quindi di automatizzare a basso costo le operazioni di campionamento e archiviazione sicura del dato.

Al fine di consentire il riutilizzo futuro di tali campioni, da parte di moduli software di terze parti, questi sono memorizzati su un database a partire dal quale è possibile recuperarli tramite precise API (Application Programming Interface).

Attualmente non è presente un sistema software in grado di utilizzare le API esposte dall'infrastruttura di raccolta e archiviazione dei campioni e soddisfare i requisiti individuati nel precedente capitolo.

E' richiesta quindi l'implementazione di un modulo software in grado di automatizzare il processo di analisi e verifica dei dati raccolti e che consenta il monitoraggio in real time e da remoto, sia rispetto alla rete di sensori che rispetto all'infrastruttura di archiviazione, delle condizioni ambientali di una specifica area.

Il modulo software da implementare deve anche integrare un componente per la segnalazione di eventi anomali eccezionali tramite e-mail.

2.2. Possibili Soluzioni

Si passa ora ad un'analisi delle possibili soluzioni analizzando i vari aspetti, prima singolarmente e poi passando a delineare la soluzione complessiva e finale.

2.2.1. Implementazione della Business Logic

Per quanto riguarda il software vero e proprio da realizzare, si hanno, allo stato attuale dell'arte, diversi linguaggi e soluzioni tecnologiche in grado di mettere lo sviluppatore nelle condizioni di giungere a una soluzione che soddisfi i requisiti richiesti.

a) Programmazione Tradizionale

Molti dei sistemi di monitoraggio ambientali esistenti utilizzano linguaggi tradizionali per l'implementazione della business logic.

Al fine di effettuare un confronto con la programmazione tradizionale, non si può non citare, tra i linguaggi maggiormente diffusi, Java. Da linguaggio nato solo per la rete, è divenuto il linguaggio di programmazione forse più utilizzato al mondo, raggiungendo un'importante diffusione sia a livello accademico, sia come piattaforma di business (i server di molte banche oggi operano su Java). Ciò fa si che utilizzando questo linguaggio è possibile realizzare pressoché qualsiasi cosa e interfacciarsi con qualunque infrastruttura esterna.

L'utilizzo di una soluzione di questo tipo richiede un'adeguata fase di progettazione delle strutture dati necessarie e delle operazioni da effettuare su di esse, seguendo il paradigma Object-Oriented. Si procede poi con la progettazione di un prodotto software in grado di sfruttare i servizi messi a disposizione dalle suddette classi e svolgere i compiti previsti in fase di analisi dei requisiti.

Questa tipologia di soluzione presenta vantaggi e svantaggi. Tra i vantaggi troviamo sicuramente:

- La larga diffusione di questo tipo di linguaggi di programmazione rende semplice reperire sviluppatori esperti nella realizzazione di sistemi software che presentino anche con un elevato grado di difficoltà;
- Trattandosi sempre di soluzioni molto utilizzate, per il committente, si ha l'ulteriore possibilità di rendere nulli i costi di formazione e, essendo alta la concorrenza tra i programmatori, rendere bassi i costi di manodopera.

Tra gli svantaggi invece troviamo:

- Nel caso di flussi di elaborazione complessi caratterizzati dall'attivazione a catena di eventi diversi, la programmazione tradizionale può rendere difficile, se non impossibile, la realizzazione e comprensione del codice;
- La presenza di codice poco chiaro e leggibile rende il committente vincolato allo specifico sviluppatore del prodotto software iniziale, in quanto risulta difficile per terzi intervenire rapidamente e in modo efficiente;
- Non tutti gli stakeholders (analisti, esperti di business, sviluppatori, etc) possono partecipare all'implementazione della business logic in quanto è richiesta una forte conoscenza specializzata;
- Nella programmazione tradizionale la business logic viene integrata e implementata insieme a tutti gli altri componenti di contorno del software, rendendone difficile l'individuazione e quindi le operazioni di aggiornamento e manutenzione;

- Inserendo la business logic all'interno del software applicativo può risultare difficile l'integrazione con regole pre-esistenti.

Come si può evincere dall'elenco sopra riportato, una soluzione che si basi sui soli linguaggi di programmazione tradizionale mal si adatta all'implementazione di un sistema software come quello in oggetto.

Infatti non risultano soddisfatti molti dei requisiti precedentemente individuati, come:

- L'indipendenza dagli altri moduli del sistema;
- La separazione dell'elaborazione degli eventi dall'archiviazione dei dati;
- La manutenibilità;
- Ecc.

b) Un Nuovo Paradigma di Programmazione

Nell'ottica della manutenibilità, del possibile ri-utilizzo e magari della modifica futura delle regole di business, si presenta la necessità di separare la modellazione delle strutture dati dalla definizione della logica di business.

Si sono così sviluppate negli ultimi anni diverse soluzioni in grado di operare in maniera più chiara e netta la separazione della business logic dal contesto applicativo, rendendola di più immediata comprensione e manutenzione.

Sul mercato sono disponibili diverse suite commerciali e open source che seguono questa nuova idea di programmazione. Una soluzione open source presenta il grande vantaggio di ridurre drasticamente i costi di licenza e utilizzo. Da non dimenticare sono poi le grandi comunities alla base di soluzioni di questo tipo, che forniscono un continuo supporto allo sviluppo e miglioramento del software.

Tra queste non si può non citare i sistemi di regole (Business Rule Management System), in particolare la soluzione open source JBoss BRMS.

JBoss è un application server open source che implementa l'intera suite di servizi Java Enterprise Edition. Essendo basato su Java, JBoss è un application server multipiattaforma, utilizzabile su qualsiasi sistema operativo. Come progetto open source, JBoss è supportato e migliorato da una enorme comunity di sviluppatori.

Concentrando il nostro focus su BRMS, esso è una piattaforma completa per la gestione delle regole di business e di elaborazione di eventi complessi. Utilizza il motore e linguaggio di regole Drools con supporto di livello enterprise per la creazione, la manutenzione e l'applicazione delle politiche aziendali di un'organizzazione, applicazione o servizio.

Le organizzazioni possono integrare logiche decisionali complesse nelle applicazioni Core Business e aggiornare rapidamente le regole di business al cambiamento delle condizioni di mercato.

JBoss BRMS consente all'organizzazione di:

- Distribuire servizi decisionali in ambienti fisici, virtuali e cloud;
- Migliorare l'agilità del business;
- Prendere velocemente decisioni coerenti ed efficienti;
- Abbreviare i cicli di sviluppo per tempi di commercializzazione.

Tra le principali caratteristiche dei motori di regole in generale e di JBoss BRMS in particolare troviamo:

- Programmazione Dichiarativa - Un motore di regole mi permette di dire “Che cosa fare” e non “Come farlo”. I programmi dichiarativi definiscono in modo esplicito soltanto lo scopo da raggiungere,

lasciando che l'implementazione dell'algoritmo sia realizzata dal software di supporto;

- I sistemi di regole possono essere utilizzati in tutti quegli scenari dove i linguaggi di programmazione "tradizionali" risultano di difficile utilizzo:
    - Quando ci sono molte regole di business.
    - Quando si necessita di essere vicini all'implementazione della business logic.
    - Quando il business necessita di essere flessibile e adattabile.
    - Quando non ci sono soluzioni "tradizionali" soddisfacenti.
    - Quando il problema risulta troppo complesso e non si conoscono algoritmi adeguati;
- Supporto alla separazione tra Dati e Logica;
- Supporto alla creazione di un repository centralizzato che costituisce la base di conoscenza del sistema;
- Aumento della Manutenibilità, Stabilità, Velocità e Scalabilità;
- Creazione di una logic "Cross Domain", quindi indipendente dalla macchina fisica o sistema operativo;
- Supporto al riutilizzo di regole pre-esistenti, oltre che semplificazione delle procedure di aggiornamento di quelle esistenti, anche se il sistema è già in esecuzione (deploy a caldo);
- Supporto di tools di integrazione. In particolare con l'ambiente di sviluppo Eclipse.
- Traduzione di tasks decisionali complessi in poche e semplici regole;

- Diversi stakeholders (analisti, esperti di business, sviluppatori, etc) possono contribuire all'implementazione della logica di business, sulla base delle loro competenze: codice Drools, Spreadsheet o Guided Editor.

Dovendo sviluppare un sistema software teso al monitoraggio di parametri ambientali, i quali possono variare anche molto velocemente in seguito al verificarsi di condizioni di anomalia (es. lo scoppio di un incendio), il principale obiettivo è massimizzare la velocità di elaborazione dei campioni e rendere semplice l'interazione tra eventi. Si predilige quindi una soluzione che sia il più possibile snella, ma allo stesso tempo di veloce implementazione e facile manutenzione e aggiornamento. L'utilizzo di un *Business Rule Management System* costituisce quindi la scelta migliore al fine di soddisfare al meglio i requisiti software.

2.3. Soluzione proposta

Per l'implementazione della business logic, si è scelto di utilizzare un sistema di regole. Oltre ai vantaggi di una soluzione di questo tipo già esposti nel paragrafo precedente, si è scelto di utilizzare JBoss BRMS perché prevede un semplice meccanismo di dipendenze basato su Maven (progetto open source, sviluppato dalla Apache, che permette di organizzare in modo molto efficiente un progetto java) che consente di integrare facilmente il sistema in un qualunque progetto di terze parti.

Avendo deciso di utilizzare la suite JBoss BRMS per l'implementazione della business logic, si prevede l'utilizzo del motore e linguaggio di regole Drools per la scrittura delle regole di business e di controllo individuate in fase di analisi.

Al fine di snellire la trattazione è stata scritta un'apposita sezione di appendice che approfondisce i principali aspetti del linguaggio Drools.

JBoss BRMS espone una serie di API (Application Programming Interface) Java che permette di interagire con il repository delle regole e processare i campioni sulla base della business logic definita.

2.4. Vantaggi derivanti dallo sviluppo del progetto

Il software introduce una serie di novità che contribuiscono al raggiungimento di una buona qualità e affidabilità del prodotto offerto al cliente finale, oltre che a una diminuzione dei tempi richiesti dal processo di monitoraggio completo.

Infatti l'architettura software sviluppata presenta tra i principali vantaggi:

- Un monitoraggio da remoto e in real-time dell'area di interesse;
- Una piena automatizzazione del processo di verifica dei campioni raccolti;
- La separazione dell'implementazione della business logic da tutto il resto del software applicativo con un evidente aumento della manutenibilità del software;
- Il possibile riutilizzo futuro del codice implementato, nonché la semplice integrazione con altre regole pre-esistenti;
- La possibilità, per stakeholders con diversi gradi di competenze, di collaborare alla realizzazione della business logic utilizzando lo strumento che gli è più congeniale.

Capitolo 3: Analisi

3.1. Casi d'uso

Il sistema software durante il suo ciclo di vita può trovarsi in uno dei due seguenti stati:

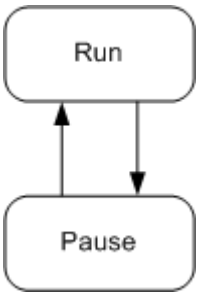

Figura 3.1: Diagramma degli stati del sistema

Uno dei principali aspetti innovativi del sistema software introdotto è sicuramente la piena automatizzazione del monitoraggio dei dati raccolti.

Ciò riduce notevolmente il numero dei casi d'uso, in quanto sarà il software, una volta avviato, ad occuparsi di svolgere tutti i compiti richiesti.

L'utente infatti interagisce direttamente con il motore di regole solo al fine di avviare o sospendere l'elaborazione di nuovi campioni.

Mentre è in esecuzione, il modulo software si deve occupare di:

- Recuperare nuovi campioni dalla base dati a intervalli di tempo regolari e pre-configurati;
- Processare i nuovi campioni sulla base della business logic richiesta;
- Nel caso si registrino eventi eccezionali, deve farsi carico di inviare tramite e-mail un'opportuna segnalazione.

L’utente può decidere in qualunque momento di sospendere l’elaborazione dei campioni. Questi continueranno comunque ad essere raccolti ed archiviati in base dati dall’infrastruttura di più basso livello.

Sfruttando l’indipendenza del nostro modulo software rispetto al mondo esterno, nel momento in cui l’utente decide di riprendere l’esecuzione, si è in grado di recuperare tutti i campioni a partire dall’ultimo processato.

Ciò consente di avere la sicurezza che la verifica dei campioni raccolti sarà sempre completa ed efficace.

In figura 3.1 si mostra il diagramma dei casi d’uso del sistema da implementare.

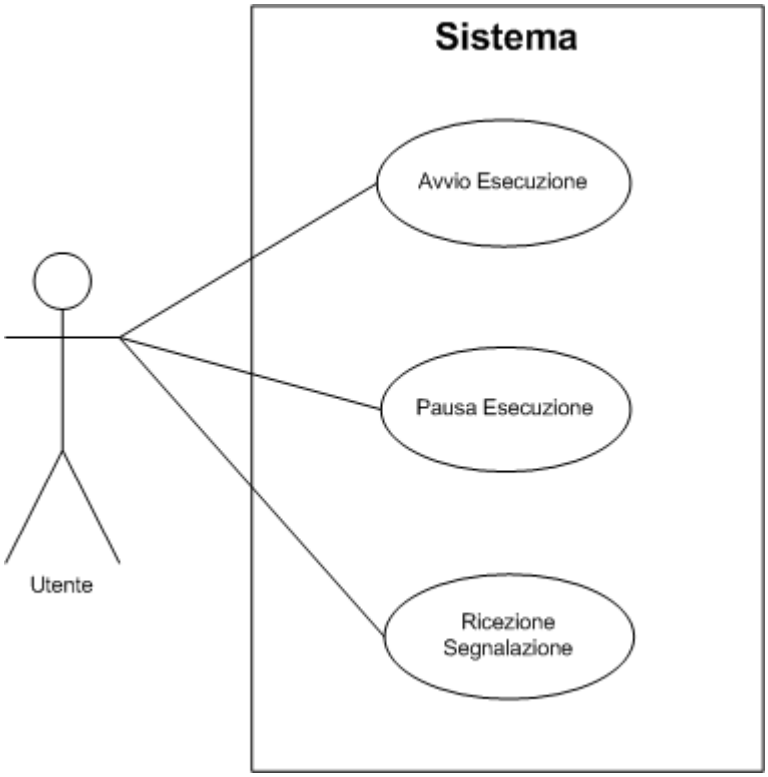

Figura 3.2: Diagramma dei casi d'uso

3.1.1. Specifiche dei casi d'uso

Si prosegue la trattazione con la descrizione dettagliata di ogni caso d'uso identificato specificando un nome, un identificatore, gli attori coinvolti, le precondizioni, il flusso principale e le post-condizioni.

Use case: **Avvio Esecuzione**
Use case ID: **UC1**
Actors: **Utente**
Preconditions: **L'utente deve dotarsi di un dispositivo in grado di accedere al modulo software**
First scenario: 1. **L'utente avvia il sistema** 2. **Il sistema inizia ad elaborare nuovi campioni in ingresso** 3. WHILE **l'utente non sospende l'elaborazione** 3.1. **Il motore preleva i nuovi dati dal database** 3.2. **Il motore processa i dati prelevati sulla base delle regole definite** IF **individuazione di un evento anomalo** 3.2.1. **Il motore invia una segnalazione per mezzo e-mail**
Postconditions: 1. **L'utente ha avviato l'esecuzione del motore di elaborazione**
Preconditions: 1. **L'utente ha precedentemente avviato l'esecuzione** 2. **L'utente ha sospeso l'attività del motore di elaborazione**
Second scenario: 1. **L'utente riavvia l'esecuzione del motore di elaborazione** 2. **Il sistema riprende l'elaborazione dei nuovi campioni in ingresso** 3. WHILE **l'utente non sospende l'elaborazione** 3.1. **Il motore preleva i nuovi dati dal database** 3.2. **Il motore processa i dati prelevati sulla base delle regole definite** IF **individuazione di un evento anomalo** 3.2.1. **Il motore invia una segnalazione per mezzo e-mail**
Postconditions: **L'utente ha riavviato l'esecuzione del motore di elaborazione**

Tabella 3.1: Caso d'uso – Avvio Esecuzione

Use case: **Pausa Esecuzione**
Use case ID: **UC2**
Actors: **Utente**
Preconditions: **L'utente ha avviato l'esecuzione del motore di regole**

Scenario: 1. **Il caso d'uso inizia quando l'utente sospende l'elaborazione** 2. **Il sistema sospende l'esecuzione del motore di regole**
Postconditions: **L'utente ha sospeso l'esecuzione del motore di elaborazione**

Tabella 3.2: Caso d'uso – Pausa Esecuzione

Use case: **Ricezione segnalazione**
Use case ID: **UC3**
Actors: **Utente**
Preconditions: **L'utente ha avviato l'esecuzione del motore di regole**
Scenario: 1. **Il motore preleva nuovi campioni dal database** 2. **Il motore processa i dati prelevati sulla base delle regole definite** 3. **Viene rilevato un evento anomalo** 4. **E' inviata una segnalazione per mezzo e-mail** 5. **L'utente riceve la segnalazione di evento anomalo** 6. **L'utente esegue le azioni previste per tale evento**
Postconditions: **L'utente ha esaminato lo stato corrente di elaborazione**

Tabella 3.3: Caso d'uso – Ricezione Segnalazione

3.2. Classi di analisi

Poiché la business logic è implementata tramite le regole Drools e quindi al di fuori delle classi come avviene, invece, nella programmazione tradizionale, è possibile realizzare semplici classi contenenti solo gli attributi di interesse, i costruttori richiesti e, per ogni attributo, i metodi *Getter* e *Setter*.

Di seguito sono presentati i package con le rispettive classi di analisi.

3.2.1. Package *Metric*

Essendo il caso d'uso relativo al monitoraggio di parametri ambientali sufficientemente critico dal punto di vista della sicurezza, si è scelto di prediligere la velocità di esecuzione rispetto all'efficienza nella gestione della memoria delle macchine fisiche.

In tale ottica, si è scelto di focalizzare l'elaborazione non su un intero messaggio ricevuto, ma su uno specifico campione in esso contenuto.

Si è quindi definito un package di classi contenente tutte le informazioni necessarie alla gestione completa di una qualunque delle tipologie di campione trasportata in uno specifico messaggio.

Ogni messaggio ricevuto, è quindi diviso in tante parti quanti sono i campioni in esso trasportati.

Il package *Metric* prevede due classi:

- *EdcMetric*, contenente tutte le informazioni relative al campione ricevuto;
- *EdcPosition*, contenente tutte le informazioni relative alla posizione geografica del sensore che ha effettuato il campionamento;

Gli attributi previsti sono di facile intuizione e corrispondono a quelli visti durante la fase di analisi del dato nel primo capitolo. Troviamo in più, nella classe *EdcMetric*, l'attributo *inControlTopic*, il quale è utilizzato per differenziare le metriche di controllo da quelle relative ai dati.

Come detto in precedenza, i metodi sono solo i costruttori necessari utilizzati per istanziare le classi e i metodi *Getter* e *Setter* per l'interazione con gli attributi delle classi.

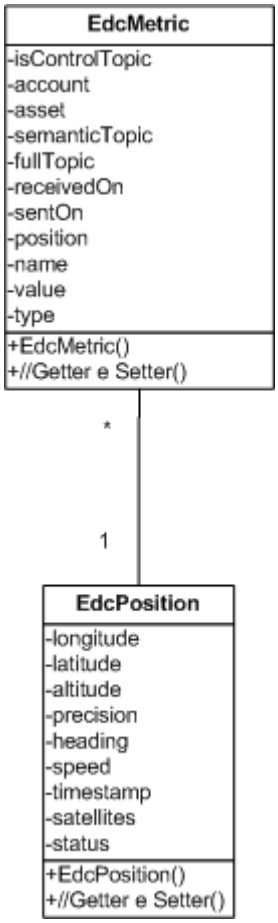

Figura 3.3: Analisi – Package Metric

Di seguito sono riportate tabelle esplicative degli attributi delle classi sopra rappresentate.

EdcMetric Class

Campi	Descrizione
isControlTopic	Campo utilizzato per differenziare una metrica di controllo da una relativa ai dati
account	Nome dell'account dell'utente
asset	Id del dispositivo della rete di sensori che ha pubblicato i dati
semanticTopic	Nome del topic su cui il campione è stato pubblicato
fullTopic	Stringa contenuta nel campo *topic* del messaggio ricevuto
receivedOn	Istante in cui il messaggio è stato ricevuto e memorizzato su database
sentOn	Istante in cui il messaggio è stato generato e inviato dal sensore
position	Posizione geografica del dispositivo
name	Nome della metrica

type	Tipologia della metrica
value	Valore della metrica

Tabella 3.4: Analisi – Classe EdcMetric

EdcPosition Class

Campi	Descrizione
longitude	Longitudine del device in gradi
latitude	Latitudine del device in gradi
altitude	Altitudine del sensore rispetto al livello del mare in metri
precision	Diluizione della precisione (DOP)
heading	Direzione del device in gradi
speed	Velocità del device in metri al secondo (m/s)
timestamp	Istante temporale letto dal sistema GPS
satellites	Numero di satelliti visti dal device
status	Stato del sistema GPS

Tabella 3.5: Analisi – Classe EdcPosition

3.2.2. Package *MetricStats*

Per ogni metrica, è necessario prevedere un package per la gestione dei dati statistici ad essa relativi.

Ognuno di tali package deve contenere le seguenti classi, dove la stringa *Metric* è sostituita con il nome della specifica metrica:

- *MetricState*, contiene informazioni statistiche assolute sulla metrica;
- *MetricThresholdState*, contiene informazioni circa il superamento delle soglie previste per tale metrica;
- *MetricHourState*, contiene informazioni statistiche relative all'ultima ora;
- *MetricDayState*, contiene informazioni statistiche relative all'ultimo giorno;
- *MetricWeekState*, contiene informazioni statistiche relative all'ultima settimana;
- *MetricMonthState*, contiene informazioni statistiche relative all'ultimo mese;

Sempre nell'ottica di massimizzare le performance, ognuna di tali classi è istanziata per ogni dispositivo e per ogni metrica.

Anche in questo caso i metodi sono:

- I costruttori necessari per istanziare le classi;
- I metodi *Getter* e *Setter* per l'interazione con gli attributi delle classi.

Di seguito è quindi esposta, per mezzo di diagrammi, la struttura dati delle classi utilizzate per lo specifico caso di studio reale in cui il software realizzato è stato sperimentato.

Alcune delle informazioni contenute in tali classi non sono direttamente utilizzate dalle regole di business oggetto di studio, ma sono comunque riportate in quanto utili ai fini di Business Analysis.

Trattandosi di strutture dati statistiche senza dipendenze reciproche, sono di seguito riportate le singole classi di analisi costituenti il package con una descrizione dei rispettivi attributi.

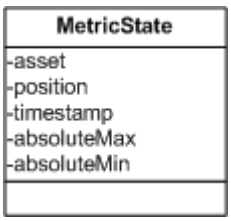

Figura 3.4: Analisi – Classe MetricState

Campi	Descrizione
asset	Id del dispositivo della rete di sensori che ha pubblicato i dati
position	Posizione geografica del dispositivo (classe definita nel package *Metric*)
timestamp	Istante dell'ultimo aggiornamento dell'oggetto
absoluteMax	Valore massimo registrato per quella metrica
absoluteMin	Valore minimo registrato per quella metrica

Tabella 3.6: Analisi – Classe MetricState

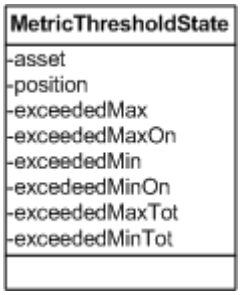

Figura 3.5: Analisi – Classe MetricThresholdState

Campi	Descrizione
asset	Id del dispositivo della rete di sensori che ha pubblicato i dati
position	Posizione geografica del dispositivo (classe definita nel package *Metric*)
exceededMax	Flag che indica se la soglia superiore per tale metrica è stata superata
exceededMaxOn	Istante in cui la soglia superiore è stata superata per l'ultima volta
exceededMin	Flag che indica se la soglia inferiore per tale metrica è stata superata
exceededMinOn	Istante in cui la soglia superiore è stata superata per l'ultima volta
exceededMaxTot	Numero totale di volte per cui la soglia superiore è stata superata
exceededMinTot	Numero totale di volte per cui la soglia inferiore è stata superata

Tabella 3.7: Analisi – Classe MetricThresholdState

Figura 3.6: Analisi – Classe MetricHourState

Campi	Descrizione
asset	Id del dispositivo della rete di sensori che ha pubblicato i dati
position	Posizione geografica del dispositivo (classe definita nel package *Metric*)
timestamp	Istante dell'ultimo aggiornamento dell'oggetto
hourMax	Valore massimo registrato per quella metrica nell'ultima ora
hourMin	Valore minimo registrato per quella metrica nell'ultima ora
hourMed	Valore medio registrato per quella metrica nell'ultima ora

hourCount	Numero di campioni registrati per quella metrica nell'ultima ora

Tabella 3.8: Analisi – Classe MetricHourState

Figura 3.7: Analisi – Classe MetricDayState

Campi	Descrizione
asset	Id del dispositivo della rete di sensori che ha pubblicato i dati
position	Posizione geografica del dispositivo (classe definita nel package *Metric*)
timestamp	Istante dell'ultimo aggiornamento dell'oggetto
dayMax	Valore massimo registrato per quella metrica nell'ultimo giorno
dayMin	Valore minimo registrato per quella metrica nell'ultimo giorno
dayMed	Valore medio registrato per quella metrica nell'ultimo giorno
dayCount	Numero di campioni registrati per quella metrica nell'ultimo giorno

Tabella 3.9: Analisi – Classe MetricDayState

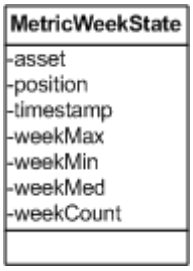

Figura 3.8: Analisi – Classe MetricWeekState

Campi	Descrizione
asset	Id del dispositivo della rete di sensori che ha pubblicato i dati
position	Posizione geografica del dispositivo (classe definita nel package *Metric*)
timestamp	Istante dell'ultimo aggiornamento dell'oggetto
weekMax	Valore massimo registrato per quella metrica nell'ultima settimana
weekMin	Valore minimo registrato per quella metrica nell'ultima settimana
weekMed	Valore medio registrato per quella metrica nell'ultima

	settimana
weekCount	Numero di campioni registrati per quella metrica nell'ultima settimana

Tabella 3.10: Analisi – Classe MetricWeekState

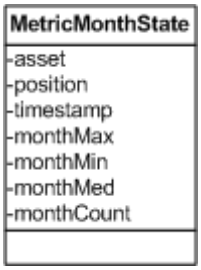

Figura 3.9: Analisi – Classe MetricMonthState

Campi	Descrizione
asset	Id del dispositivo della rete di sensori che ha pubblicato i dati
position	Posizione geografica del dispositivo (classe definita nel package *Metric*)
timestamp	Istante dell'ultimo aggiornamento dell'oggetto
monthMax	Valore massimo registrato per quella metrica nell'ultimo mese
monthMin	Valore minimo registrato per quella metrica nell'ultimo mese
monthMed	Valore medio registrato per quella metrica nell'ultimo mese
monthCount	Numero di campioni registrati per quella metrica nell'ultimo mese

Tabella 3.11: Analisi – Classe MetricMonthState

3.3. Regole

Durante la fase di analisi dei requisiti funzionali, la logica del sistema software da implementare è stata suddivisa la in tre diversi macro-gruppi:

- Logica di monitoraggio a breve termine;
- Logica di monitoraggio a lungo termine;
- Logica di controllo dell'infrastruttura hardware di raccolta e archiviazione dei campioni.

Si procede quindi con un'analisi approfondita dei tre blocchi, facendo riferimento allo specifico caso di studio reale in cui il software realizzato è stato sperimentato.

Ciò ha portato all'individuazione delle specifiche regole che costituiscono la logica di business.

3.3.1. Logica di business a breve termine

Nel breve periodo il software deve rilevare condizioni ambientali di anomalia verificatesi in un piccolo lasso temporale, rappresentato dall'ultimo minuto.

Le condizioni di anomalia da tenere sotto controllo sono le seguenti:

- Per ciascuna metrica sono previsti valori rientranti in un certo intervallo, definito sulla base dell'area di installazione della rete di sensori. Al di fuori di questo, sia superiormente che inferiormente, è necessario verificare il corretto funzionamento del dispositivo e che non si siano presentate condizioni anomale. Tali soglie, in relazione alle corrispondenti unità di misura, sono riportate per ciascuna metrica nella seguente tabella:

Metriche	Minimo	Massimo
E	-130	+25
PRESS	1	100
PM10	0	50
NO2	0	200
CO2	0	1500
CO	0	10000
O3	0	180
VOC	0	50
TEMP	-20	+50
HUM	20	80

Tabella 3.12: Soglie

- Un incremento della temperatura dell'aria fa diminuire l'umidità relativa della stessa e favorisce il verificarsi di incendi. E' quindi

necessario monitorare temperatura e umidità in relazione al loro valore medio dell'ultima ora. In particolare, un nuovo campione della temperatura deve avere un valore non superiore al 10% del valore medio orario e allo stesso tempo quello di umidità non inferiore al 10%.

- Un incremento delle diverse tipologie di inquinanti presenti nell'aria (tra quelli monitorati: PM10, VO2, CO2, CO, O3, VOC) può avere effetti negativi di vario genere sia sulla salute umana che sull'ambiente circostante. E' quindi necessario che i nuovi campioni non superino del 20% il valore medio dell'ultimo giorno.
- In condizioni di alta umidità e temperatura, l'aria risulta molto pesante. Di conseguenza la soglia indicata al punto precedente si abbassa al 10%.
- Anche i valori del campo elettromagnetico devono essere monitorati, al fine di tenere sotto controllo l'inquinamento elettromagnetico o *elettrosmog*. Questo valore non deve differire del 20% rispetto al valore medio orario.

In seguito al verificarsi di una qualsiasi di queste anomalie è previsto l'invio di un'opportuna notifica alle autorità competenti.

3.3.2. Logica di business a lungo termine

Per quanto riguarda invece il monitoraggio nel lungo periodo, questo consiste in un'attività di reportistica tesa ad aggregare i dati in ingresso ottenendo report su base oraria, giornaliera, settimanale, mensile.

Periodicamente è quindi prevista l'esecuzione di un gruppo di regole che si occupa di recuperare dal database i campioni relativi al periodo di interesse e aggiornare opportunamente le statistiche.

Effettuato l’aggiornamento dei diversi reports, è poi possibile metterli in relazione tra loro e individuare peggioramenti delle condizioni ambientali generali intervenendo rapidamente, al fine di adottare misure preventive adeguate:

- La concentrazione media settimanale di CO2 nell’aria, ha subito una variazione superiore al 10% rispetto alla stessa concentrazione su base mensile. Questa regola va implementata seguendo la stessa logica anche per le altre tipologie di inquinanti monitorati (PM10, VO2, CO, O3, VOC).
- L’umidità e la temperatura media su base oraria sono rispettivamente del 10% e del 5% superiori agli stessi valori su base giornaliera. In questo caso la soglia di tolleranza espressa al primo punto scende al 5%.
- Le variazioni di campo elettromagnetico su base giornaliera e su base settimanale sono rispettivamente superiori al 10% e al 5% rispetto alla media mensile.

Anche in questo caso, in seguito al verificarsi di una qualsiasi di queste condizioni è previsto l'invio di un'opportuna notifica alle autorità competenti.

3.3.3. Logica di controllo

Accanto alla logica di business vera e propria è prevista anche una logica di controllo dell’infrastruttura in grado di rilevare e segnalare eventuali problemi hardware occorsi:

- Un dispositivo si disconnette inaspettatamente e non si riconnette in un tempo utile di un minuto.

- Non sono pubblicati nuovi campioni di una data metrica per un periodo di tempo superiore a 5 minuti.

Anche in questo caso, in seguito al verificarsi di una qualsiasi di queste condizioni è previsto l'invio di un'opportuna notifica alle autorità competenti.

3.4. Diagrammi di sequenza

Di seguito sono riportati i diagrammi di sequenza relativi ai principali casi d'uso individuati.

Per quanto riguarda l'avvio dell'elaborazione l'utente si occuperà di dialogare con il *"Gestore del Motore di regole"* comunicandogli la volontà di avviare il processo di analisi di nuovi dati.

Da questo punto in poi l'esecuzione è completamente controllata dal software.

Tale modulo si occupa quindi di:

- Interrogare il database a intervalli di tempo prefissati richiedendo nuovi campioni;
- Inserire i nuovi dati nella sessione corrente del motore di regole;
- Avviare l'esecuzione di tutte le regole configurate;
- Ricevere il log, esaminarlo e inviare, nel caso in cui si osserva la presenza di un'anomalia, un'apposita segnalazione tramite e-mail all'utente.

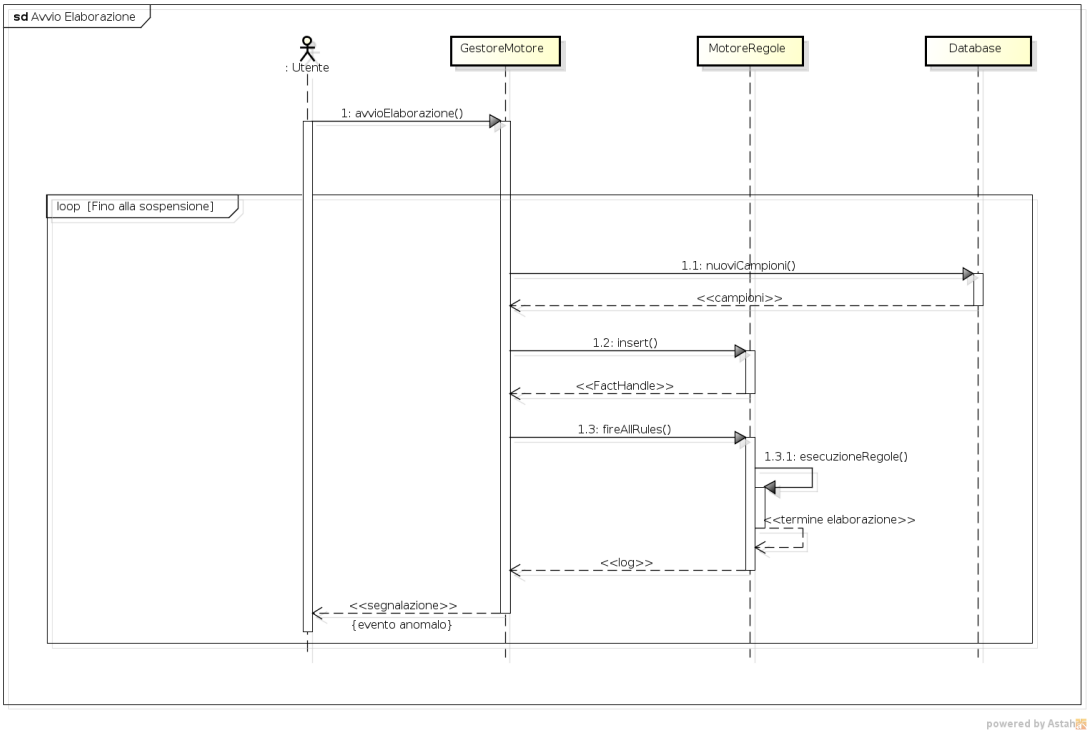

Figura 3.10: Analisi – Avvio Elaborazione

L'utente può decidere in qualunque momento di sospendere l'elaborazione di nuovi campioni.

Anche in questo caso, egli comunica al *"Gestore del Motore di regole"* la volontà di sospendere l'esecuzione. A sua volta questo modulo inoltrerà la segnalazione al motore delle regole.

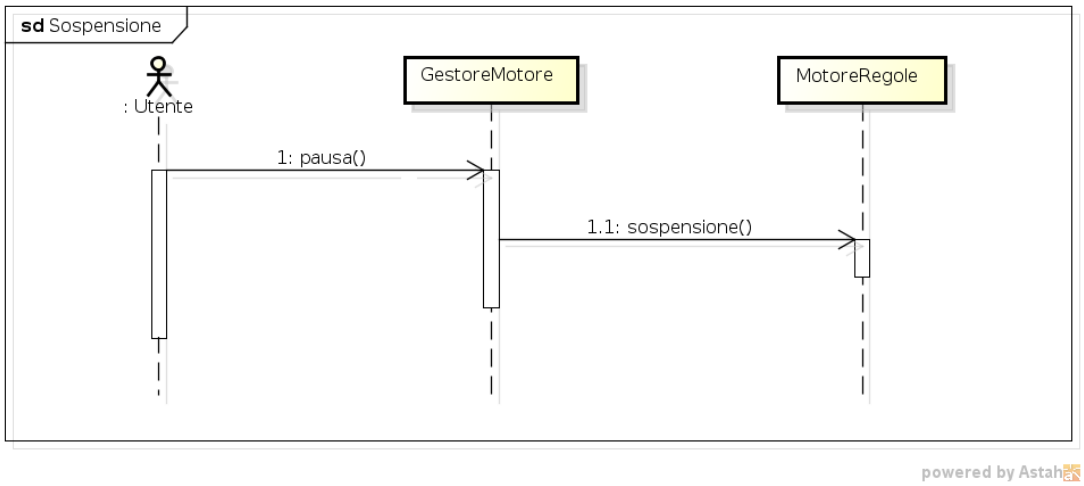

Figura 3.11: Analisi – Sospensione

Capitolo 4: Progettazione

4.1. Classi di progetto

Si procede completando la specifica delle classi individuate in fase di analisi.

4.1.1. Package *Metric*

Il package *Metric* è utilizzato per la gestione dei singoli campioni ambientali ricevuti dalla rete di sensori.

Il package prevede tutti gli attributi e metodi necessari per la corretta ricezione, memorizzazione ed elaborazione dei dati ricevuti.

Per completare la descrizione delle informazioni e funzionalità offerte (es. la lettura dei singoli campioni), sono stati indicanti parametri e valori restituiti da tutti i metodi previsti. Anche gli attributi sono stati ulteriormente specificati indicandone il tipo.

Per quanto riguarda la semantica dei metodi e degli attributi definiti, resta valido quanto detto durante la fase di analisi.

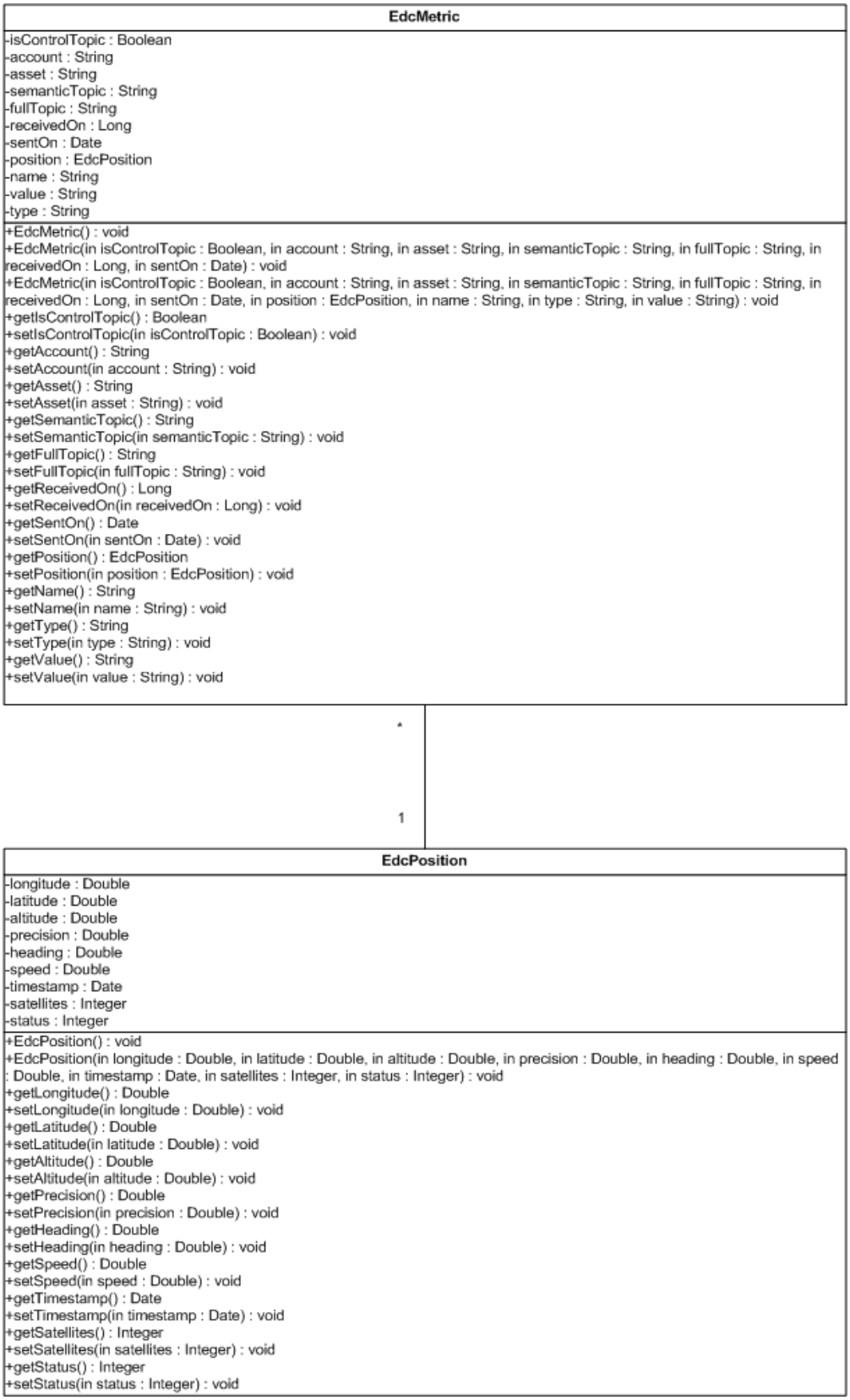

Figura 4.1: Progetto – Package Metric

4.1.2. Package *MetricStats*

Il package *MetricStats* è destinato alla gestione dei dati statistici ottenuti aggregando i campioni ricevuti dalla rete di sensori.

Per ogni dispositivo e per ogni metrica sono quindi calcolate e memorizzate statistiche su base oraria, giornaliera, settimanale, mensile. A queste si aggiunge una struttura dati contenente informazioni circa il superamento delle soglie previste per tale metrica.

Anche in questo caso è stata completata la descrizione delle informazioni e delle funzionalità offerte, indicando i tipi degli attributi, dei parametri e dei valori restituiti dai metodi previsti.

Il package contiene ora tutti gli attributi e i metodi necessari per la corretta elaborazione e memorizzazione dei dati statistici*.

Per quanto riguarda la semantica dei metodi e degli attributi definiti, resta valido quanto detto durante la fase di analisi.

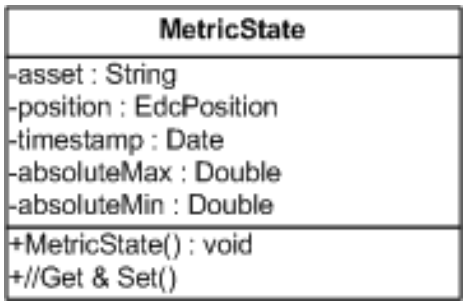

Figura 4.2: Progetto – Classe MetricState

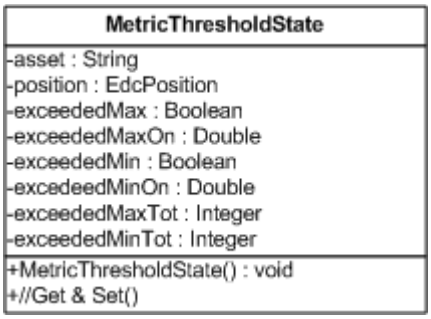

Figura 4.3: Progetto – Classe MetricThresholdState

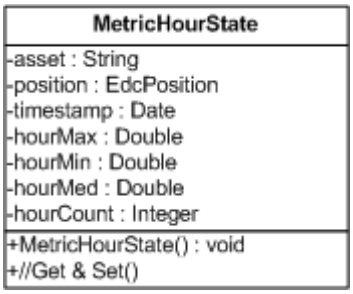

Figura 4.4: Progetto – Classe MetricHourState

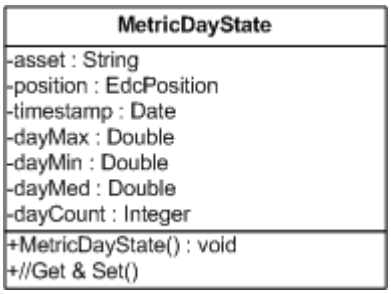

Figura 4.5: Progetto – Classe MetricDayState

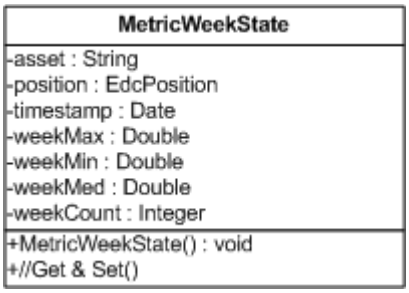

Figura 4.6: Progetto – Classe MetricWeekState

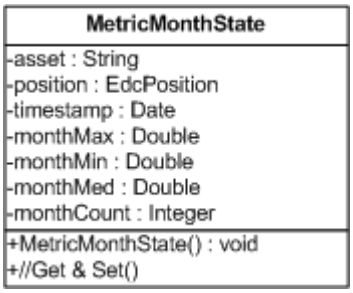

Figura 4.7: Progetto – Classe MetricMonthState

Note:

* Per rendere più agevole la lettura e la comprensione dei diagrammi, in molti casi sono state sottintese le operazione di settaggio dei vari parametri e i corrispondenti valori restituiti.

4.2. Diagrammi di sequenza

Di seguito sono riportati i diagrammi di sequenza relativi ai principali casi d'uso individuati.

Anche in questo caso, ne è stata completata la descrizione indicando i tipi degli attributi, dei parametri e dei valori restituiti dai metodi chiamati.

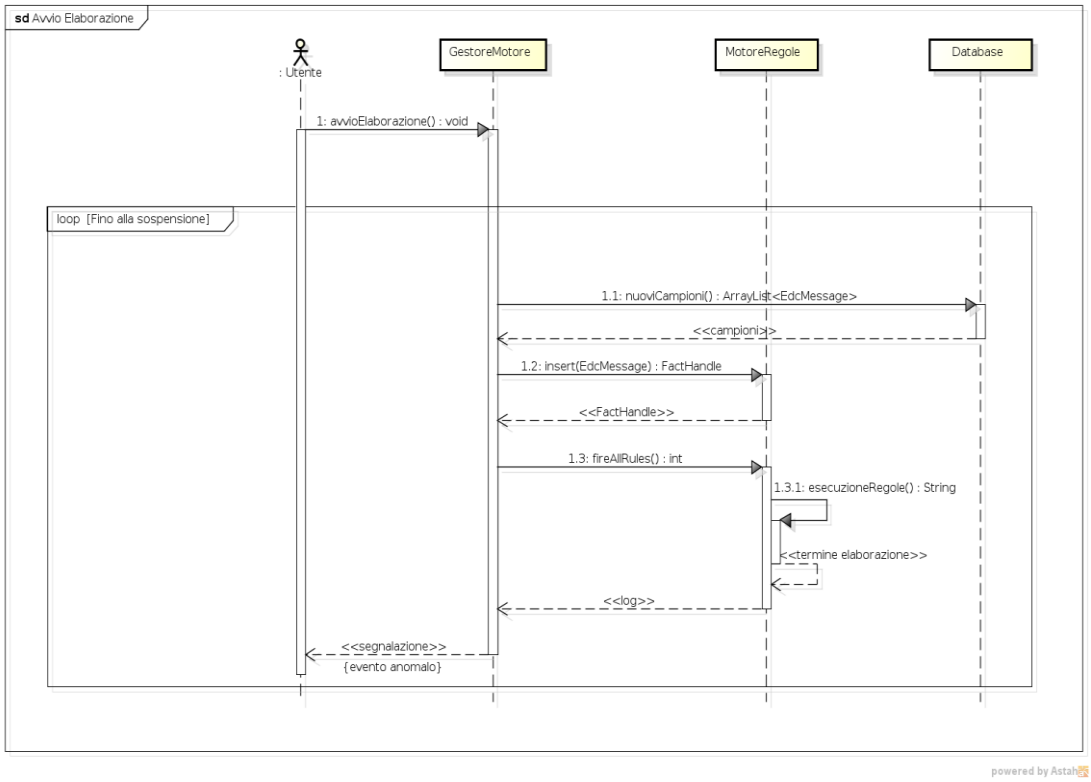

Figura 4.8: Progetto – Avvio Elaborazione

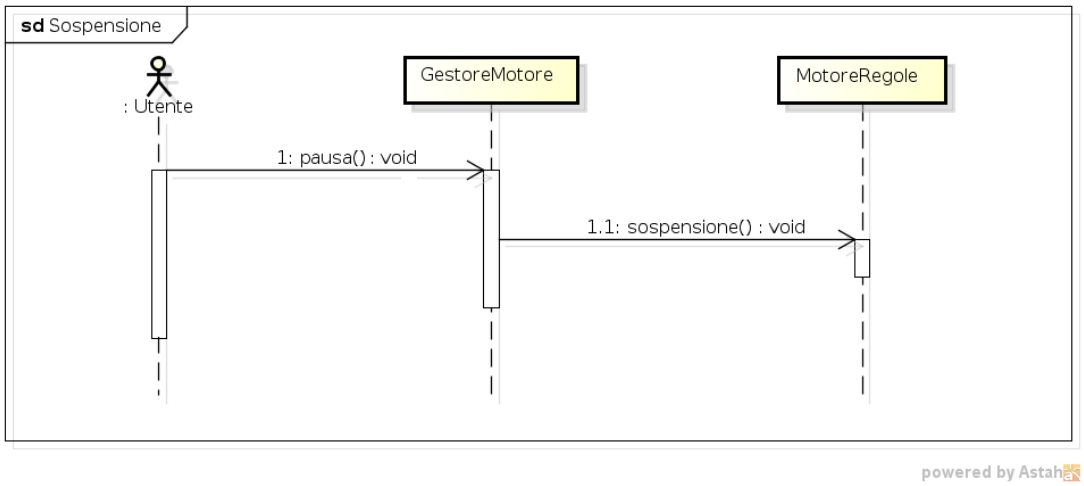

Figura 4.9: Progetto – Sospensione

Capitolo 5: Implementazione

JBoss BRMS sfrutta il motore e linguaggio di regole *Drools* per la scrittura, la gestione e l'esecuzione delle regole sia di business che di controllo.

JBoss BRMS espone inoltre una serie di *API Java* che permettono di interagire con il repository delle regole e processare i campioni sulla base della business logic definita.

L'utilizzo di JBoss BRMS come unità centralizzata di scrittura, conservazione, gestione ed esecuzione delle regole porta con sé una serie di vantaggi.

Tra questi non si possono non citare:

- La capacità di tradurre flussi di elaborazione complessi in poche e semplici regole;
- La semplicità di apprendimento e utilizzo dei singoli costrutti del linguaggio Drools;
- La completa separazione tra business logic e qualsiasi logica applicativa;
- La riusabilità di regole pre-esistenti;
- La semplicità delle operazioni di modifica e aggiornamento delle singole regole, anche senza la necessità di fermare l'ambiente di esecuzione (deploy a caldo);
- Possibilità per qualunque modulo di terze parti di interagire con il motore realizzato sfruttando le API esposte;

- La possibilità per diversi stakeholders (analisti, esperti di business, sviluppatori, etc) di collaborare, sulla base delle proprie conoscenze, all'implementazione della business logic tramite gli strumenti che più gli sono congeniali: GUI Editor, fogli elettronici, codice Drools, ecc.

5.1. Gestore del motore delle regole

Come si è potuto evincere dai diagrammi di sequenza precedentemente esposti, il motore di regole presenta uno strato *"cuscinetto"*, costituito da un suo modulo gestore il quale svolge fondamentalmente due compiti:

- Prendere in consegna le richieste di avvio e sospensione dell'esecuzione dal mondo esterno e comunicarle al motore vero e proprio;
- Ad elaborazione attiva, prelevare periodicamente nuovi campioni dal database, inserirli nella sessione del motore di regole e avviare la loro elaborazione secondo la business logic definita.

Il codice implementato, relativo alla gestione vera e propria del motore di regole, è di seguito riportato:

```
package it.extrasys.eurotech.business;

import it.extrasys.eurotech.javabean.AuthenticationError;
import it.extrasys.eurotech.javabean.Log;
import it.extrasys.eurotech.javabean.Messages;
import it.extrasys.eurotech.scheduler.BrmsScheduler;

import java.util.ArrayList;
import java.util.Date;

import org.kie.api.KieServices;
import org.kie.api.builder.ReleaseId;
import org.kie.api.runtime.KieContainer;
import org.kie.api.runtime.KieSession;
import org.quartz.SchedulerException;
import org.springframework.beans.factory.annotation.Autowired;
```

```
import org.springframework.context.ApplicationContext;
import org.springframework.stereotype.Component;

import com.eurotech.cloud.apis.v2.model.EdcMessage;
import com.eurotech.cloud.apis.v2.model.EdcPayload;
import com.sun.jersey.api.client.Client;

import org.acme.eurotech.modeltwo.EdcMetric;
import org.acme.eurotech.modeltwo.EdcPosition;
import org.drools.compiler.kproject.ReleaseIdImpl;

@Component
public class BrmsBusiness {

    // >>>>>> Set these variables according to your Cloud user account
    public static final String API_URL = "https://api-sandbox.everyware-cloud.com/v2/";    //URL for API
    private String USERNAME;                    //Username in account, requires Administrative permissions
    private String PASSWORD;                    // Password associated with Username
    // <<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<<

    static Client              client    = null;

    static ArrayList<EdcMetric> reals = new ArrayList<EdcMetric>();

    @Autowired
    private ApplicationContext applicationContext;
    @Autowired
    private BrmsScheduler myScheduler;
    @Autowired
    private Log log;
    @Autowired
    private Messages messages;
    @Autowired
    private AuthenticationError authenticationError;
    private KieSession ksession1;

    public String getUSERNAME() {
            return this.USERNAME;
    }

    public void setUSERNAME(String username) {
            this.USERNAME = username;
    }

    public String getPASSWORD() {
            return this.PASSWORD;
    }

    public void setPASSWORD(String password) {
            this.PASSWORD = password;
    }

    public boolean initBrms() throws Exception {
        //Authentication
        messages.updateMessages(this.USERNAME, this.PASSWORD);
        if ((authenticationError.getError()).equals("Authorization failed")) {
            return false;
        }

        //Creation Ksession
        KieServices ks = KieServices.Factory.get();
        ReleaseId releaseId = new ReleaseIdImpl("org.acme.eurotech","rule-one","2.0");
```

```
        KieContainer kcontainer = ks.newKieContainer(releaseId);
        ksession1 = kcontainer.newKieSession("KSession1");

        //Settings relating to the control of the time of sending mails
        Date date = new Date();
        ksession1.setGlobal("periodSend", 120);
        ksession1.setGlobal("tempSend", date.getTime()-(6000*1000)-1);
        ksession1.setGlobal("co2Send", date.getTime()-(6000*1000)-1);
        ksession1.setGlobal("inactiveSend", date.getTime()-(6000*1000)-1);
        ksession1.setGlobal("anyDeviceMissingSend", date.getTime()-(6000*1000)-1);
        ksession1.setGlobal("singleDeviceMissingSend", date.getTime()-(6000*1000)-1);

        //Update Ksession and fire the rules
        updateKsession(messages.getMessages());

        return true;
    }

    //Scheduler activation
    public void startBrms() throws SchedulerException{
        myScheduler.resumeAllTriggers();
    }

    //Read new messages and fire the rules
    public void execute() throws SchedulerException {
            messages.updateMessages(this.USERNAME, this.PASSWORD);
                    updateKsession(messages.getMessages());
    }

    //Update Ksession and fire the rules
    private void updateKsession( ArrayList<EdcMessage> edcMessages ) {
            if (edcMessages != null) {
                    for (EdcMessage edcMessage : edcMessages) {
                    boolean isControlTopic = true;

                String account = edcMessage.getTopic().substring(0, edcMessage.getTopic().indexOf("/"));
                String asset = edcMessage.getTopic().substring(edcMessage.getTopic().indexOf("/")+1,
edcMessage.getTopic().indexOf("/", edcMessage.getTopic().indexOf("/")+1));
                String semanticTopic = edcMessage.getTopic().substring(edcMessage.getTopic().indexOf("/",
edcMessage.getTopic().indexOf("/")+1)+1);
                String fullTopic = edcMessage.getTopic();

                Long receivedOn = edcMessage.getTimestamp().getTime();

                EdcPayload edcPayload = edcMessage.getEdcPayload();
                Date sentOn = edcPayload.getTimestamp();
                EdcPosition position;
                if ( edcPayload.getPosition() != null ) {
                   position = new EdcPosition(

            edcPayload.getPosition().getLongitude(),

            edcPayload.getPosition().getLatitude(),

            edcPayload.getPosition().getAltitude(),

            edcPayload.getPosition().getPrecision(),

            edcPayload.getPosition().getHeading(),

            edcPayload.getPosition().getSpeed(),
```

```
                edcPayload.getPosition().getTimestamp(),
                edcPayload.getPosition().getSatellites(),
                edcPayload.getPosition().getStatus()
                                                                                    );
                        }
                        else {
                           position = new EdcPosition();
                        }

                        if ( edcPayload.getMetrics().getMetrics() != null ) {
                           for ( com.eurotech.cloud.apis.v2.model.EdcMetric m : edcPayload.getMetrics().getMetrics() )
{
                                    EdcMetric edcMetric = new EdcMetric( isControlTopic, account, asset,
semanticTopic, fullTopic, receivedOn, sentOn, position, m.getName(), m.getType(), m.getValue());
                                    ksession1.insert(edcMetric);
                           }
                        }
                        else {
                           EdcMetric edcMetric = new EdcMetric( isControlTopic, account, asset, semanticTopic,
fullTopic, receivedOn, sentOn );
                           ksession1.insert(edcMetric);
                        }
                     }
                     ksession1.fireAllRules();
                     log.aggiorna();
                  }
   }

   //Scheduler stand-by
   public void pauseBrms() throws SchedulerException {
            myScheduler.pauseAllTriggers();
            //ksession.dispose();
   }
}
```

Dopo i metodi *set* e *get* relativi ai parametri *username* e *password*, utilizzati rispettivamente per il settaggio e la lettura dei valori necessari all'autenticazione dell'utente, troviamo il metodo *initBrms*.

Tale metodo si occupa di:

1. Effettuare l'autenticazione presso il database;
2. Recuperare il repository delle regole e istanziare la *ksession* del motore di regole *Drools*;
3. Settare le variabili temporali relative all'invio delle mail di allerta;
4. Richiamare il metodo *updateKsession*.

Tale metodo esamina l'ultimo messaggio letto dal database e, per ogni metrica in esso contenuta, aggiorna la *ksession* e avvia l'esecuzione delle regole, aggiornando contestualmente il log stampato a video dall'applicazione web.

Troviamo poi i metodi *startBrms* e *pauseBrms* utilizzati dal controller Spring per gestire la sospensione e la riattivazione del motore. Nel dettaglio, questi metodi attivano e disattivano rispettivamente uno *scheduler*, il quale, tramite un *trigger*, recupera un nuovo messaggio dalla base dati e richiama il metodo *updateKsession*.

Di seguito è invece riportato il semplice codice java relativo a *scheduler* e *trigger* di esecuzione.

```
package it.extrasys.eurotech.scheduler;

import org.quartz.SchedulerException;
import org.springframework.scheduling.quartz.SchedulerFactoryBean;

public class BrmsScheduler extends SchedulerFactoryBean {

	@Override
	public void afterPropertiesSet() throws Exception {
		// TODO Auto-generated method stub
		super.afterPropertiesSet();
		pauseAllTriggers();
	}

	public void pauseAllTriggers() throws SchedulerException {
		getScheduler().pauseAll();
	}

	public void resumeAllTriggers() throws SchedulerException {
		getScheduler().resumeAll();
	}
}


package it.extrasys.eurotech.scheduler;

import it.extrasys.eurotech.business.BrmsBusiness;

import org.quartz.SchedulerException;
import org.springframework.beans.factory.annotation.Autowired;

public class BrmsTrigger {

	@Autowired
	BrmsBusiness brmsBusiness;

	public void runJob() throws SchedulerException{
```

```
                brmsBusiness.execute();
        }
}
```

La prima classe contiene i metodi necessari all'attivazione e disattivazione dei trigger, mentre la seconda, il trigger vero e proprio, richiama il metodo *execute* della classe *brmsBusiness*, precedentemente analizzata.

5.2. Logica di business a breve termine

Nei sotto-paragrafi successivi è riportato il codice Drools relativo alle regole individuate in fase di analisi e implementate nel corso del progetto di tesi.

5.2.1. Regola 1

Per ciascuna metrica sono previsti valori rientranti in un certo intervallo, definito sulla base dell'area di installazione della rete di sensori. Al di fuori di questo, sia superiormente che inferiormente, è necessario verificare il corretto funzionamento del dispositivo e che non si siano presentate condizioni anomale. Tali soglie, in relazione alle corrispondenti unità di misura, sono riportate per ciascuna metrica nella seguente tabella:

Metriche	Minimo	Massimo
E	-130	+25
PRESS	1	100
PM10	0	50
NO2	0	200
CO2	0	1500
CO	0	10000
O3	0	180
VOC	0	50
TEMP	-20	+50
HUM	20	80

Tabella 5.1: Soglie

Al fine di semplificare l'esposizione viene presentata la regola solo per il parametro CO2.

Dalla regola di seguito riportata, è facile osservare come, con poche e semplici istruzioni, la parte *when* della regola si occupa di calcolare una media sugli ultimi 60 secondi dei valori pubblicati per quella metrica sul topic indicato dallo specifico dispositivo. Se tale valore è al di fuori dell'intervallo previsto per tale metrica, viene eseguita la parte *then*. Quest'ultima si occupa di:

1. Comporre destinatario, oggetto e corpo della mail;
2. Inviare la mail, tramite servizio REST;
3. Aggiornare il file di log;
4. Scrivere su un file CSV il dato che ha attivato la regola così da poterlo riutilizzare in futuro per eventuali lavori statistici;
5. Aggiornare la console del software.

Da notare è l'inserimento nel codice di una *query*, utile al recupero dell'ultimo dato pubblicato sul topic e che ha causato l'attivazione della regola. Tale query è utilizzata nella parte java del codice, in quanto qui non abbiamo accesso diretto ai dati presenti nella *ksession* Drools.

```
package org.acme.eurotech.test_one_rule;

query getLastCo2
	lastCo2: EdcMetric( asset == "00:E0:C7:09:36:AD", semanticTopic == "ems/data", name == "CO2" )
over window:length(1)
end

rule "Co2-Metric_Threshold"

  when

	total : Number( doubleValue > 1500 )
		from accumulate( EdcMetric ( asset == "00:E0:C7:09:36:AD", semanticTopic == "ems/data",
name == "CO2", v : Double.parseDouble(value) ) over window:time(60s)
	    ,
	    average( v )
	)

	or

	total : Number( doubleValue < 0 )
```

```
                    from accumulate( EdcMetric ( asset == "00:E0:C7:09:36:AD", semanticTopic == "ems/data",
name == "CO2", v : Double.parseDouble(value) ) over window:time(60s)
                ,
                average( v )
            )

    then

            //Recupero l'ultimo dato pubblicato sul topic e compongo destinatario, oggetto e corpo della mail
            String to = "antonino.cosenza@extrasys.it";
            String body = "";
            String object = "OGGETTO: "+kcontext.getRule().getName();
            QueryResults results = kcontext.getKieRuntime().getQueryResults("getLastCo2");
            Iterator<QueryResultsRow> iterator = results.iterator();
            if (iterator.hasNext()) {
                    QueryResultsRow result = iterator.next();
                    EdcMetric lastEdcMetric = (EdcMetric) result.get("lastCo2");
                    body = "On "+new Date(lastEdcMetric.getReceivedOn())+", device
"+lastEdcMetric.getAsset()+" published on topic "+lastEdcMetric.getSemanticTopic()+" an average CO2 value
of "+total.toString()+" for the last 1 minute.";
            }

            if (!body.equalsIgnoreCase("")) {
                //Scrivo sul file di Log
                try {
                    PrintWriter out = new PrintWriter(new BufferedWriter(new
FileWriter("/home/antonino/myfile.txt", true)));
                    out.println(new Date()+" fired rule " + kcontext.getRule().getName());
                    out.println("Average: "+total);
                    out.close();

                    //Scrive sul file di csv
                    out = new PrintWriter(new BufferedWriter(new FileWriter("/var/www/Eurotech/myfile.csv",
true)));
                    SimpleDateFormat DATE_FORMAT = new SimpleDateFormat("dd/MM/yyyy HH:mm:ss");
                    out.println(DATE_FORMAT.format(new Date())+";;"+total);
                    out.close();
                } catch (Exception e) {
                    e.printStackTrace();
                }

                    //Scrivo sulla console
                    System.out.println("fired rule " + kcontext.getRule().getName());
                    System.out.println("Average: "+total);

                    //Chiamata REST per richiedere l'invio della mail di alert
                    try {
                      URL url = new URL("http://localhost:9001/Eurotech-mail");
                      HttpURLConnection connection = (HttpURLConnection) url.openConnection();
                      connection.setRequestProperty("Content-Type", "text/plain; charset=utf-8");

                      //Invio l'oggetto e il corpo della mail come corpo della POST
                      connection.setDoOutput(true);           //abilita la scrittura
                      connection.setRequestMethod("POST");          //settaggio del metodo POST
                      OutputStreamWriter wr = new OutputStreamWriter(connection.getOutputStream());
                      wr.write(to+"/&"+object+"/&"+body);             //scrittura del content
                      wr.flush();

                      //Invio la richiesta
                      while (connection.getResponseCode() != 200) ;
                      System.out.println(kcontext.getRule().getName()+": mail inviata");
                    }
```

```
                    catch ( Exception e ) {
                     e.printStackTrace();
                    }
            }
end
```

Per le regole successive, la parte *then* conserva un'analoga struttura; verrà quindi esposta e discussa solo la parte *when* al fine di semplificare la trattazione.

5.2.2. Regola 2

Un incremento della temperatura dell'aria fa diminuire l'umidità relativa della stessa e favorisce il verificarsi di incendi. E' quindi necessario monitorare temperatura e umidità in relazione al loro valore medio dell'ultima ora. In particolare, un nuovo campione della temperatura deve avere un valore non superiore al 10% del valore medio orario e allo stesso tempo quello di umidità non inferiore al 10%.

Ogni qualvolta viene letta una nuova metrica relativa alla temperatura e all'umidità, la regola si occupa di:

1. Recuperare le ultime metriche di temperatura e umidità lette dal database;
2. Recuperare le corrispondenti strutture dati statistiche su base oraria relative al dispositivo che ha pubblicato il campione;
3. Verificare la condizione di anomalia sopra riportata.

```
rule "Temp-Hum-Metric_Threshold"

   when

            metricTemp : EdcMetric ( asset == "00:E0:C7:09:36:AD", semanticTopic == "ems/data", name ==
"TEMP", temp : Double.parseDouble(value) )

            metricHum : EdcMetric ( asset == "00:E0:C7:09:36:AD", semanticTopic == "ems/data", name ==
"HUM", hum : Double.parseDouble(value) )

            tempHourState: TempHourState ( asset == metricTemp.getAsset() )
```

```
        humHourState: HumHourState ( asset == metricHum.getAsset() )
        Number( (tempHourState.hourMed + (tempHourState. hourMed /100*10)) < temp)
        Number( (humHourState.hourMed - (humHourState. hourMed /100*10)) > hum)
```

5.2.3. Regola 3

Un incremento delle diverse tipologie di inquinanti presenti nell'aria (tra quelli monitorati: PM10, VO2, CO2, CO, O3, VOC) può avere effetti negativi di vario genere sia sulla salute umana che sull'ambiente circostante. E' quindi necessario che i nuovi campioni non superino del 20% il valore medio dell'ultimo giorno.

Anche in questo caso è presentata la regola relativa alla sola CO2 per motivi di trattazione.

Ogni qualvolta viene letta una nuova metrica relativa alla Co2, la regola si occupa di:

1. Recuperare l'ultima metrica di Co2 letta dal database;
2. Recuperare le corrispondenti strutture dati statistiche su base giornaliera relative al dispositivo che ha pubblicato il campione;
3. Verificare la condizione di anomalia sopra riportata.

```
rule "Co2-Day-Metric"
  when
        metricCo2 : EdcMetric ( asset == "00:E0:C7:09:36:AD", semanticTopic == "ems/data", name == "CO2", co2 : Double.parseDouble(value) )
        co2DayState: Co2DayState ( asset == metricCo2.getAsset() )
        Number( (co2DayState.dayMed - (co2DayState.dayMed /100*20)) < co2)
```

5.2.4. Regola 4

In condizioni di alta umidità e temperatura, l'aria risulta molto pesante. Di conseguenza la soglia indicata al punto precedente si abbassa al 10%.

In questo caso è semplicemente necessario formulare, per ognuna delle regole previste nel paragrafo 5.2.3, una corrispondente regola dove, oltre ad abbassare la percentuale calcolata per la Co2 al valore del 10%, sono aggiunte le due seguenti condizioni:

```
metricTemp : EdcMetric ( asset == "00:E0:C7:09:36:AD", semanticTopic == "ems/data", name == "TEMP", temp : Double.parseDouble(value)  > 30)

metricHum : EdcMetric ( asset == "00:E0:C7:09:36:AD", semanticTopic == "ems/data", name == "HUM", hum : Double.parseDouble(value)  > 70 )
```

Tali istruzioni si occupano di recuperare dalla sessione i nuovi valori campionati e assicurarsi che rispettino le soglie stabilite.

5.2.5. Regola 5

Anche i valori del campo elettromagnetico devono essere monitorati, al fine di tenere sotto controllo l'inquinamento elettromagnetico o *elettrosmog*. Questo valore non deve differire del 20% rispetto al valore medio orario.

```
rule "E-Hour-Metric"

  when

metricE : EdcMetric ( asset == "00:E0:C7:09:36:AD", semanticTopic == "ems/data", name == "E", e : Double.parseDouble(value) )

eHourState: EHourState ( asset == metricE.getAsset() )

Number((eHourState.hourMed - (eHourState.hourMed /100*20)) > e || (eHourState.hourMed - (eHourState.hourMed /100*20)) < e)
```

Le tre istruzioni sopra riportate si occupano rispettivamente di:

- Recuperare dalla sessione l'ultima metrica campionata con *name == "E"* e salvarne il valore dopo un'opportuna conversione di tipo;
- Recuperare dalla sessione la struttura dati statistica richiesta;

- Verificare il rispetto della soglia.

5.3. Logica di business a lungo termine

Per quanto riguarda invece il monitoraggio nel lungo periodo, questo consiste in un'attività di reportistica tesa ad aggregare i dati in ingresso ottenendo report su base oraria, giornaliera, settimanale, mensile.

Anche in questo caso, viene inviata una mail di allerta ogni qualvolta è riscontrata una condizione di anomalia.

5.3.1. Regola di aggiornamento delle strutture dati statistiche

Periodicamente è prevista l'esecuzione di un gruppo di regole incaricate di recuperare dal database i campioni relativi al periodo di interesse e aggiornare opportunamente le statistiche.

Le regole implementate si occupano di:

1. Prelevare tutti i nuovi campioni registrati nell'ultima ora;
2. Recuperare la struttura dati relativa;
3. Effettuare il suo opportuno aggiornamento.

Di seguito è riportata la regola relativa all'aggiornamento della struttura dati *Co2HourState*.

```
package org.acme.prova.prova_one;

rule "Co2-UpdateHourly"
when

	metric: EdcMetric ( asset == "00:E0:C7:09:36:AD", name == "CO2", v : Double.parseDouble(value) , timestamp : receivedOn) over window:time(1h)
	co2HourState: Co2HourState ( asset == metric.getAsset() )

then

	//Scrivo sulla console
```

```
        System.out.println("fired rule: " + kcontext.getRule().getName());

        //Aggiorno la struttura dati
        modify (co2HourState) { setTimestamp (timestamp) };
        If (v > co2HourState.getHourMax())
                modify (co2HourState) { setHourMax (v) };
        If (v < co2HourState.getHourMin())
                modify (co2HourState) { setHourMin (v) };
        modify (co2HourState) { setHourMed ((getHourMed()*getHourCount()  + v) / (getHourCount() + 1))
};
        modify (co2HourState) { setHourCount (getHourCount() + 1) };

end
```

5.3.2. Regola 1

La concentrazione media settimanale di CO2 nell'aria, ha subito una variazione superiore al 10% rispetto alla stessa concentrazione su base mensile. Questa regola va implementata seguendo la stessa logica anche per le altre tipologie di inquinanti monitorati (PM10, VO2, CO, O3, VOC).

La regola, in seguito a un aggiornamento delle strutture dati statistiche su base settimanale e mensile relative alla Co2 in corrispondenza del dispositivo indicato, si comporta nel seguente modo:

1. Recupera le strutture dati statistiche richieste;
2. Verifica la condizione di cui sopra.

```
rule "Co2-Week-Month-Metric"

  when

        co2WeekState: Co2WeekState ( asset == "00:E0:C7:09:36:AD", co2WeekMed : weekMed )
        co2MonthState: Co2MonthState ( asset == "00:E0:C7:09:36:AD", co2MonthMed : monthMed )

        Number((co2MonthMed - (co2MonthMed /100*10)) < co2WeekMed)
```

5.3.3. Regola 2

L'umidità e la temperatura media su base oraria sono rispettivamente del 10% e del 5% superiori agli stessi valori su base giornaliera. In questo caso la soglia di tolleranza espressa al primo punto scende al 5%.

Anche in questo caso è necessario aggiungere una nuova regola a partire da quella indicata nel paragrafo precedente, modificando la soglia come indicato e aggiungendo le seguenti condizioni:

```
        tempHourState: TempHourState ( asset == "00:E0:C7:09:36:AD" )
        humHourState: HumHourState ( asset == "00:E0:C7:09:36:AD" )
        tempDayState: TempDayState ( asset == "00:E0:C7:09:36:AD" )
        humDayState: HumDayState ( asset == "00:E0:C7:09:36:AD" )
        Number( (tempDayState.hourMed + (tempDayState.hourMed /100*10)) < tempHourState.dayMed)
        Number( (humDayState.hourMed + (humDayState.hourMed /100*5)) < humHourState.dayMed)
```

Le prime quattro istruzioni si occupano di recuperare dalla sessione le strutture dati richieste, mentre le altre due di effettuare i controlli opportuni sulle nuove soglie.

5.3.4. Regola 3

Le variazioni di campo elettromagnetico su base giornaliera e su base settimanale sono rispettivamente superiori al 10% e al 5% rispetto alla media mensile.

La regola, in seguito a un aggiornamento delle strutture dati statistiche su base giornaliera, settimanale e mensile relative al campo elettromagnetico in corrispondenza del dispositivo indicato, si comporta nel seguente modo:

1. Recupera le strutture dati statistiche richieste;
2. Verifica le condizioni di cui sopra.

```
rule "Co2-Week-Month-Metric"
  when
        eDayState: EDayState ( asset == "00:E0:C7:09:36:AD", eDayMed : dayMed )
        eWeekState: EWeekState ( asset == "00:E0:C7:09:36:AD", eWeekMed : weekMed )
        eMonthState: EMonthState ( asset == "00:E0:C7:09:36:AD", eMonthMed : monthMed )
        Number((eMonthMed - (eMonthMed /100*10)) < eDayMed)
```

```
        or
        Number((eMonthMed - (eMonthMed /100*5)) < eWeekMed)
```

Anche in questo caso, le prime tre istruzioni si occupano di recuperare dalla sessione le strutture dati richieste, mentre le altre di effettuare i controlli secondo la logica definita.

Da notare come la presenza del costrutto *or* faccia sì che il motore di regole Drools, crei in memoria due regole separate una per ognuna delle due condizioni.

5.4. Logica di controllo

Accanto alla logica di business vera e propria è prevista anche una logica di controllo dell'infrastruttura in grado di rilevare e segnalare eventuali problemi hardware occorsi.

5.4.1. Regola 1

Un dispositivo si disconnette inaspettatamente e non si riconnette in un tempo utile di un minuto.

La regola sotto riportata verifica che nel caso in cui un dispositivo si disconnetta inviando un messaggio sul topic *"MQTT/LWT"*, nei successivi 60 secondi invii anche un messaggio di riconnessione sul topic *"MQTT/BIRTH"*. In caso contrario, sfruttando la parte *then* della regola, invia un'apposita segnalazione come visto in precedenza.

```
rule "Single_Device_Missing"
   when
                a : EdcMetric( isControlTopic == true, semanticTopic == "MQTT/LWT",
asset=="00:E0:C7:09:36:AD" ) over window:time(60s)
                not EdcMetric( isControlTopic == true, semanticTopic == "MQTT/BIRTH", asset == a.asset,
this after a )
```

```
  then
```

Tale regola ha però il difetto di richiedere una copia per ogni singolo dispositivo, in quanto è presente anche una condizione di controllo sull'id.

Una versione più generica ed unica della regola precedente, valida per tutti i dispositivi è la seguente:

```
rule "Any_Device_Missing"
  when
                    a : EdcMetric( isControlTopic == true, semanticTopic == "MQTT/LWT" ) over window:time(60s)
                    not EdcMetric( isControlTopic == true, semanticTopic == "MQTT/BIRTH", asset == a.asset, this after a )
  then
```

In questo caso ci si è semplicemente limitati ad eliminare la verifica dell'id del device.

5.4.2. Regola 2

Non sono pubblicati nuovi campioni di una data metrica per un periodo di tempo superiore a 5 minuti.

```
rule "Topic_Inactive"
  when
    not EdcMetric( asset == "00:E0:C7:09:36:AD", semanticTopic == "ems/data" ) over window:time(5m)
  then
```

Da notare come la regola, sfruttando la flessibilità e la leggibilità del codice Drools, risulta essere molto semplice e chiara.

Capitolo 6: Conclusioni e sviluppi futuri

6.1. Innovazioni introdotte

Il sistema software realizzato ha consentito di sfruttare le API esposte dall'infrastruttura di raccolta e archiviazione dei campioni ambientali, introducendo così notevoli vantaggi dal punto di vista del controllo ambientale e meteorologico.

E' infatti possibile, in real time e in maniera completamente automatizzata, riuscire a rilevare sia eventi anomali verificatisi nel breve periodo sia un peggioramento generale delle condizioni di inquinamento dell'aree di interesse.

L'utilizzo di un architettura basata su singole regole e flussi di regole, ha consentito di creare un componente software semplice, ma allo stesso tempo molto efficiente e veloce nell'analisi dei dati sulla base della business logic definita.

La possibilità, offerta da JBoss BRMS, di creare un repository centralizzato delle regole, lontano dalla logica applicativa, e di esporlo al mondo esterno consente, a qualunque altro componente di terze parti che voglia riutilizzare tali regole, di importarle e utilizzarle in modo semplice e veloce, nel pieno rispetto del principio di riutilizzo del codice.

La struttura basata su un punto centrale di gestione delle regole, unito con la separazione tra definizione e utilizzo del codice, consente un'alta manutenibilità del codice. Infatti qualunque stakeholders, può intervenire andando ad aggiornare la definizione della business logic al fine di adattarla ai

cambiamenti dei requisiti, in maniera completamente indipendente dai moduli che la stanno utilizzando.

Un'ulteriore vantaggio offerto dalla piattaforma di JBoss BRMS è sicuramente quello di portare finalmente la definizione della business logic al di fuori dei software applicativi.

Ciò consente anche agli addetti ai lavori (analisti, esperti di business, etc), i quali avranno forti competenze di settore ma magari scarse doti informatiche, di partecipare all'individuazione e definizione della business logic sfruttando gli strumenti che gli sono più congeniali: GUI Editor, fogli elettronici, codice Drools, ecc.

6.2. Sviluppi Futuri

La definizione della business logic tramite un sistema di regole e flussi di regole indipendenti dagli moduli utilizzatori del codice, apre molti scenario di utilizzo da parte di applicazioni di terze parti.

Con la sempre maggiore diffusione della rete internet e delle connessioni a banda larga cablate o wireless, si è registrata negli ultimi anni una esponenziale diffusione di dispositivi portatili.

Si pensi quindi banalmente alla creazione di un'applicazione web, realizzata tramite uno dei molteplici framework MVC (model-view-controller) di integrazione delle pagine web con codice applicativo java e quindi in grado di sfruttare le API esposte da JBoss BRMS, che consenta tramite un semplice smartphone di monitorare da remoto lo stato del sistema.

Da semplice applicazione web di monitoraggio dello stato attuale del sistema, si potrebbero poi integrare molte funzionalità, quali la gestione e modifica della business logic o la realizzazione di tabelle e grafici 3D a partire

dai dati raccolti, tale da farla diventare una vera e propria piattaforma di controllo ambientale.

Un altro aspetto da estende è rappresentato dal fatto che allo stato attuale di sviluppo, il software consente di inviare segnalazioni del verificarsi di eventi anomali, esclusivamente tramite e-mail. Si potrebbe quindi estendere tale capacità applicativa, andando a integrare l'invio di segnalazione sia tramite i tradizionali SMS, che tramite i moderni social network.

Appendice 1: Drools

I sistemi di regole forniscono un nuovo modo per l'implementazione della business logic, tramite l'utilizzo di appositi linguaggi di regole, come Drools, invece dei linguaggi procedurali tradizionali.

Una soluzione di questo tipo porta con sé una serie di importanti vantaggi, già discussi nel secondo capitolo, tra i quali merita sicuramente di essere ricordata la separazione della business logic dal contesto applicativo.

Rules

Ogni regola è costituita da due parti:

- una parte "condizioni" indicata dalla keyword "when" e chiamata LHS (Left Hand Side);
- una parte "azioni" indicata dalla keyword "then" e chiamata RHS (Right Hand Side);.

Le regole lavorano sui fatti (i dati), che nel nostro caso sono le istanze degli oggetti applicativi. Questi ultimi devono essere dei semplici JavaBean contenenti i descrittori e i metodi set e get per ogni attributo della classe.

Quando tutte le condizioni sono verificate, la regola è attivata.

Di seguito lo scheletro di una regola:

```
rule "<name>"
<attribute> <value>
when
      <conditions>
then
      <consequence>
end
```

Il nome della regola può essere costituito anche da più parole. In questo caso esso deve essere racchiuso tra apici.

Di seguito troviamo poi gli attributi della regola nel formato nome-valore, utili per il controllo dell'esecuzione della regola stessa. Essi sono:

```
salience <int>
agenda-group <string>
no-loop <boolean>
auto-focus <boolean>
duration <long>
ruleflow-group <string>
```

Abbiamo quindi la parte LHS della regola, che consiste in una serie di vincoli sui fatti.

Ogni vincolo è generalmente costituito da una serie di condizioni sugli attributi di uno specifico tipo di oggetto. Tutti gli oggetti di quel tipo che soddisfano tutte le condizioni espresse, verificano il vincolo.

Nella parte LHS, nel caso di più vincoli, è implicita la presenza del connettore "AND". E' possibile inserire il connettore "OR" che porterà internamente alla generazione di due regole separate, una per ogni condizione espressa. In questo modo, se le condizioni non sono mutualmente esclusive la regola può essere attivata più volte.

Subito dopo troviamo la parte RHS, costituita da un qualunque JavaCode.

Tipiche azioni che sono realizzate in questa parte della regola sono: inserimento, modifica o cancellazione di un fatto; modifica di un campo di un fatto; modifica del valore di una global; aggiunta di una global alla working memory.

Di seguito le API utilizzate per eseguire le suddette azioni:

- `update(fact);`
- `insert(new Fact());`
- `retract(fact);`
- `modify(fact) {<expression> [ , <expression> ]}`
- `insertLogical(new Fact());`

Quando un oggetto viene inserito in working memory tramite l'API insert(), viene restituito il suo FactHandle. Questo rappresenta l'oggetto

inserito e può essere utilizzato per interagire con la working memory, modificando o eliminando l'oggetto.

File DRL

Il file .drl è il file contenente le regole definite.

Esso presenta la seguente struttura:

- Package: il package a cui le regole definite dal file appartengono;
- Expander: lista dei file .dsl utilizzati;
- Import: lista di file contenenti definizioni di tipi e oggetti di dominio importanti all'interno del nostro package in stile java. Le istanze di tali oggetti, i fatti, saranno inseriti a runtime in working memory;
- Global: sono comunemente sopranominate "variabili" e i suoi valori sono inseriti in working memory (attraverso l'API setGlobal). Sono utilizzate per restituire risultati e per riferire dati, ma devono essere usate con cura nella parte LHS delle regole. Questo perché ,a differenza dei fatti, una loro modifica non viene valutata in automatico dal motore di regole;
- Function: sono blocchi di codice java utilizzato all'interno di una "eval" o espressione sia nel LHS, che nel RHS.
- Rule: le regole vere e proprie per le quali il file è stato creato.

This

La keyword "this" è uno speciale auto-riferimento all'oggetto stesso per il quale si sta esprimendo una condizione. Su di esso è possibile utilizzare ogni operatore: ==, !=, memberOf, contains, etc, in maniera appropriata.

Il suo utilizzo risulta molto utile quando si vuole comparare l'istanza di un fatto con un altro dello stesso tipo, escludendo il fatto stesso:

```
rule “find people who are the same age”
when
      p1 : Person()
      p2 : Person( this != p1, age == p1.age )
then
      # add to results list
end
```

Costrutti avanzati

Al fine di rendere Drools un potente linguaggio e motore di regole sono stati introdotti dei costrutti di programmazione avanzati:

• Exists: elemento condizionale valutato “true” quando esiste almeno un fatto che soddisfa tutte le condizioni espresse.

```
rule "Item Out of Stock"
when
      customer: Customer()
      order: Order(customerId == customer.customerId)
      exists OrderItem(orderId == order.orderId,
itemStatus == “out_of_stock”)
then
      order.setStatus(“out-of-stock”);
end
```

• Not: elemento condizionale valutato “true” quando non esiste nessun fatto che soddisfa tutte le condizioni espresse.

```
rule "Items in Stock"
when
      customer: Customer()
      order: OrderHeader(customerId == customer.customerId)
      not OrderItem( orderId == order.orderId,
itemStatus == “out_of_stock” )
then
      order.setStatus(“items in stock”);
end
```

• From: questo costrutto permette di specificare una sorgente dati esterna alla working memory. Questa può essere costituita semplicemente dai risultati di una funzione o dall’interazione con un altro componente dell’applicazione o da dati persistenti recuperati tramite query da un database.

```
rule "Apply discount for large orders by gold customers”
when
      order : Order()
      Item(custId : customerId == order.customerId, quantity >
10)
```

```
      Customer(status == “gold”) from
hibernateSession.getNamedQuery("Get Customer“)
.setProperties({customerId = custId})
then
      order.setDiscount(.10);
end
```

• Forall: elemento condizionale valutato “true” quando tutti i fatti che soddisfano la prima condizione, soddisfano anche le rimanenti condizioni.

```
rule "All english buses are red"
when
      forall( bus : Bus( type == “english”)
      Bus( this == bus, color == “red” ) )
then
      # all english buses are red
end
```

• Collect: consente alle regole di lavorare su collezioni di oggetti provenienti dalla working memory o da una sorgente dati esterna. Il valore restituito è sempre una classe concreta che implementa l’interfaccia java.util.Collection e che fornisce un costruttore di default senza argomenti. Il costrutto accetta sorgenti nidificate e un esempio di un suo utilizzo reale è il seguente:

```
import java.util.LinkedList;
rule "Send a message to all mothers"
when
      town : Town( name == “Paris” )
      mothers : LinkedList() from collect( Person( gender ==
“F”, children > 0 ) from town.getPeople() )
then
      # send a message to all mothers
end
```

• Accumulate: trattandosi di un costrutto molto potente e complesso, ad esso è stato riservato un intero paragrafo.

Altri operatori utilizzabili per esprimere le condizioni di un vincolo sono:

• matches

• contains

• not matches

- not contains
- in
- not in
- memberOf
- not memberOf

Accumulate

Questo costrutto è una versione più flessibile e potente del costrutto *collect*. In più rispetto a quest'ultimo, consente a una regola di iterare sugli oggetti di una collezione, eseguendo azioni custom e restituendo come risultato un oggetto. La sua sintassi generale è la seguente:

```
<result pattern> from accumulate ( <source pattern>,
            init( <init code> ),
            action( <action code> ),
            reverse( <reverse code> ),
            result( <result expression> ) )
```

`<source pattern>`: vincolo che esprime una serie di condizioni sugli oggetti provenienti dalla sorgente dati.

`<init code>`: blocco di codice java eseguito una volta prima delle iterazioni sulla sorgente dati.

`<action code>`: blocco di codice java eseguito per ogni oggetto proveniente dalla sorgente dati.

`<reverse code>`: blocco di codice java opzionale eseguito per ogni oggetto proveniente dalla sorgente dati che non soddisfa più il vincolo espresso dal `<source pattern>`. Lo scopo di questo codice è effettuare l'"undo" di ogni azione effettuata dall'`<action code>`.

`<result expression>`: blocco di codice java eseguito dopo l'iterazione sugli oggetti della sorgente dati.

<result pattern>: vincolo che esprime una serie di condizioni sul risultato dell'*accumulate*. Se il vincolo è soddisfatto, si procede con la valutazione degli altri vincoli del LHS, altrimenti la regola è considerata "non attivata".

```
rule "Apply 10% discount to orders over US$ 100,000"
when
      order1 : Order()
      orderTotal : Number( doubleValue > 100000 ) from
      accumulate( OrderItem( order == order1, ovalue : value
      ),
            init( double total = 0; ),
            action( total += ovalue; ),
            reverse( total -= ovalue; ),
            result( total ) )
then
      # apply discount to order1
end
```

Questo costrutto diventa ancora più potente quando è utilizzato insieme ad alcune funzioni note come *Accumulate Functions*:

- Average
- Min
- Max
- Count
- Sum

```
rule "Apply 10% discount to orders over US$ 100,000"
when
      order1 : Order()
      total : Number( doubleValue > 100000 ) from accumulate(
OrderItem( order == order1, ovalue : value ),
            sum( ovalue ) )
then
      # apply discount to order1
end
```

Controllo dell'esecuzione delle regole

Dopo aver inserito un nuovo fatto in working memory o averne modificato uno già presente tramite il suo FactHandle, si può impartire al motore l'ordine di valutare ed eseguire le regole, tramite l'API *fireAllRules()* applicata alla sessione corrente.

Ciò comporta la valutazione di tutte le regole che vengono così inserite in *Agenda*, una sorta di contenitore dove risiedono tutte le regole

attive. Viene selezionata una di essa ed eseguita. Dopo l'esecuzione di ogni regola, viene ricalcolata l'agenda delle regole da eseguire. Nel caso di cattiva programmazione si può così facilmente cadere in un loop, essendo attivata sempre la stessa regola o gruppo di regole.

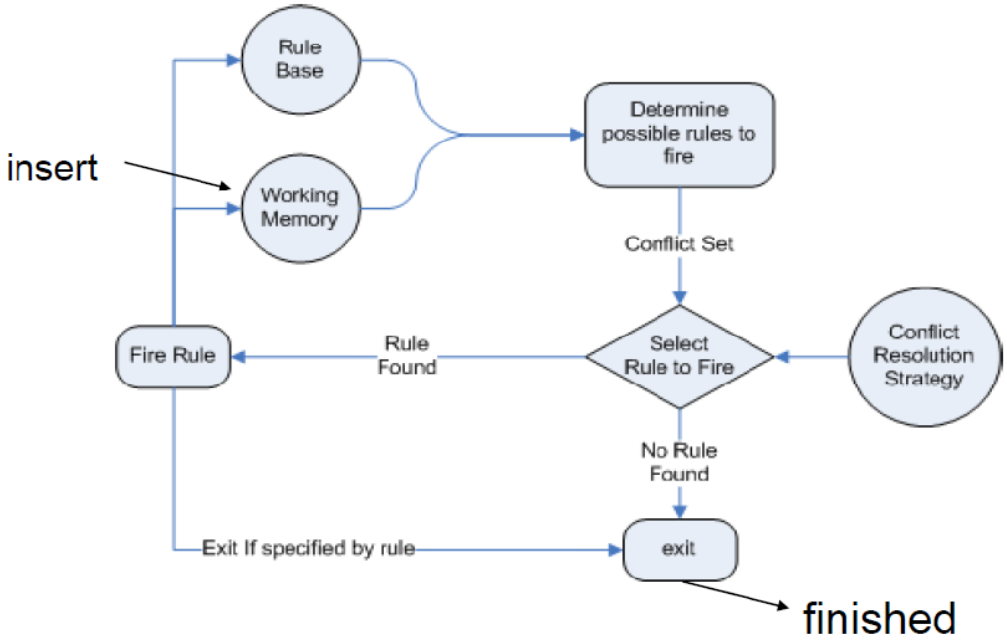

Figura Appendice 0.1: Flusso di Esecuzione

Nasce quindi l'esigenza di meccanismi in grado di consentirci un controllo completo dell'esecuzione delle regole definite.

Uno dei più efficaci strumenti a nostra disposizione è rappresentato sicuramente dagli attributi delle regole stesse.

Quando due regole sono attivate, il motore ha la necessità di selezionarne una ed eseguirla. Nell'effettuare tale scelta vengono seguite due strategie:

- FIFO (First In First Out): secondo la quale la prima delle due regole inserite in agenda, è eseguita per prima;
- Utilizzando l'attributo s*alience* impostato per tutte le regole. Questo attributo accetta come parametro dei valori interi, anche negativi.

Un valore maggiore di questo attributo in corrispondenza di una regola, indica la precedenza di quella regola rispetto alle altre. Il suo valore di default è 0.

Tuttavia nel caso di molte regole, l'utilizzo esclusivo di queste due strategie rende difficile una piena gestione dell'ordine di esecuzione delle regole.

A questo scopo ci viene incontro l'attributo *agenda-group*. Esso permette di partizionare le regole in diverse sotto-agende di quella root, sempre presente. E' quindi possibile dare il focus a una specifica sotto-agenda, alle cui regole si vuole assegnare una maggiore priorità. Se una regola non è assegnata esplicitamente a nessuna agenda, apparterrà all'agenda *root*.

Il focus a una regola viene dato tramite una specifica API, *setFocus()*, applicata all'*agenda-group* oppure tramite l'attributo *auto-focus* applicato alle regole di quella specifica *agenda-group*.

```
rule "my rule"
agenda-group "groupname"
auto-focus true
when
...
then
...
end
```

Quando tutte le regole dell'agenda che ha il focus sono state eseguite, questo passa all'agenda successiva nello stack di esecuzione e così via finché non vengono eseguite tutte le regole attivate.

Oltre al concetto di *agenda-group*, da ricordare è anche quello di *activation-group*. In questo caso, di tutte le regole di un *activation-group*, solo la prima attivata è eseguita, mentre le altre sono scartate a priori.

Drools offre poi la possibilità di associare a una regola un timer. Dopo un certo intervallo di tempo, se tutte le condizioni di quella regola sono vere,

la regola è attivata ed eseguita. E' possibile specificare anche un intervallo di ripetizione.

```
rule "Silver Priority"
timer (int: 30s 5m)
when
      customer : Customer( subscription == "Silver" )
      ticket : Ticket( customer == customer, status == "New" )
then
      ticket.setStatus( "Escalate" );
      update( ticket );
end
```

Al fine di risolvere automaticamente le condizioni di cattiva programmazione, è possibile evitare il loop dell'esecuzione di una specifica regola tramite l'attributo *no-loop*. Impostando questo attributo a *true*, in corrispondenza di una regola, questa non potrà mai ri-attivare se stessa modificando lo stato della working memory.

Ma la riattivazione di una regola può avvenire anche ad opera di altre regole. Per evitare la condizione di loop, in questo caso ci viene in aiuto l'attributo *lock-on-active*. Questo impedisce che regole di una specifica agenda-group possano riattivare altre regole della stessa agenda-group.

```
rule "my rule"
agenda-group "groupname"
lock-on-active true
when
...
then
...
end
```

Complex Event Processing

Un evento è un significativo cambiamento di stato in un particolare istante nel tempo.

Complex Event Processing, o CEP, si occupa di elaborare tutti gli eventi con lo scopo di individuare quelli più significativi e svolgere specifiche azioni sulla base di questi.

CEP utilizza tecniche come modelli complessi di eventi, correlazione e astrazione di eventi, gerarchie di eventi, relazioni tra eventi come la causalità, l'appartenenza e la tempistica, e processi event-driven.

Alcune tra le caratteristiche principali degli eventi in tale ambito sono:

- Grande quantità di eventi, ma solo pochi di reale interesse;
- Generalmente gli eventi sono immutabili;
- Generalmente le regole sono eseguite in maniera reattiva;
- Forte relazione temporale tra eventi;
- I singoli eventi sono generalmente poco importanti;
- La composizione e aggregazione di eventi è più importante.

Gli eventi non sono altro che fatti con alcune caratteristiche particolari:

- Hanno un tempo di vita gestibile;
- Di solito sono immutabili;
- Hanno forti vincoli temporali;
- Permettono l'uso di finestre temporali.

Quindi tutti gli eventi sono fatti, ma non tutti i fatti sono eventi.

Al fine di introdurre il concetto di tempo e poter lavorare con gli eventi è necessario eseguire il motore di regole in modalità *Stream*.

La sintassi per la dichiarazione di un fatto come evento è la seguente:

```
Declare eventName
      @role (event)
      @timestamp (timestamp)
      @duration (duration)
end
```

Una volta dichiarati gli eventi, è possibile metterli in relazione tramite opportuni operatori: *before*, *after*, *during*, *starts*, *finishes*, *includes*, *etc*.

Gli eventi permettono poi di lavorare su finestre mobili. Queste possono dipendere sia da un intervallo temporale che dal numero di eventi da includere nella finestra.

Il vincolo di una finestra temporale sugli ultimi 60 secondi si esprime nella parte LHS della regola tramite la sintassi:

```
over window:time(60s)
```

Allo stesso modo, una finestra che include gli ultimi 30 eventi si esprime con:

```
over window:length(30)
```

Indice delle figure

Indice delle tabelle

Bibliografia

[1] K. Desai, D. Hoffman, D. Le Sage. *"Red Hat JBoss BRMS 6.0 Development Guide"*. Red Hat Engineering Content Services, (2014)

[2] D. Hoffman, E. Kopalova , J. Wulf. "*Red Hat JBoss BRMS 6.0 Administration And Configuration Guide".* Red Hat Content Services (2014).

[3] JBoss Drools team. *"Drools Documentation"*. Red Hat.

[4] K. Desai, D. Hoffman, E. Kopalova. "*Red Hat JBoss BRMS 6.0 User Guide".* Red Hat Content Services, (2014).

[5] Hans-Erik Eriksson, Magnus Penker. *"Business Modeling with UML, Business Patterns at Work"*. Wiley Computer Publishing.

[6] B. Mozaffari. *"Business Process Management with Red Hat JBoss BPM Suite 6".* Red Hat (March 2014)

Ringraziamenti

Un doveroso e sentito ringraziamento va al Prof. Mario Giovanni C.A. Cimino, che mi ha dato la possibilità di inserirmi fin da subito nel mondo lavorativo; ringrazio Pasquale Idà che mi ha seguito durante l'intero lavoro di Tesi e mi ha sostenuto con pazienza, disponibilità e professionalità; ringrazio la Prof.ssa Gigliola Vaglini per avermi dato l'opportunità di lavorare a questo progetto rendendomi più competente di quanto non fossi all'inizio.

Ringrazio i miei genitori e mia sorella Mariacarla che mi hanno sostenuto e incoraggiato durante tutto il periodo di studio.

Non posso non ringraziare i miei due compagni di avventure, Vincenzo e Diego. Il primo perché altrimenti non avrei tante storie divertenti da raccontare e il secondo per aver condiviso con me tanti momenti di studio e non.

Ringrazio infine tutte quelle persone che ho conosciuto in questi anni e che hanno contribuito a farmi diventare la persona che sono oggi.